2022

Towards More Efficient Collaborative Distributed Data Analysis and Learning

Zixia Liu
University of Central Florida

TOWARDS MORE EFFICIENT COLLABORATIVE DISTRIBUTED DATA ANALYSIS AND LEARNING

by

ZIXIA LIU
M.A. University of Kansas, 2012
M.S. Jilin University, 2009
B.S. Jilin University, 2007

A dissertation submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy
in the Department of Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Spring Term
2022

Major Professor: Liqiang Wang

ABSTRACT

Modern information era gives rise to the persistent generation of large amounts of data with rapid speed and broad geographical distribution. Obtaining knowledge and understanding via analysis and learning from such data have invaluable worth. Features of such data analytical tasks commonly include: data can be large scale and geographically distributed; computing capability demand can be enormous; tasks can be time-critical; some data can be private; participants can have heterogeneous capabilities and non-IID data; and multiple simultaneously submitted data analytical tasks can be possible. These bring challenges to contemporary computing infrastructure and learning models.

In view of this, we develop techniques with the purpose of tackling above challenges together towards more efficient collaborative distributed data analysis and learning. We propose a hierarchical framework that supports data analytics on multiple Apache Spark clusters. We propose reinforcement learning based resource management approaches to improve overall efficiency and reduce deadline violations for scheduling general and time-critical data analytical workflows among computing resources. We establish a new hybrid framework for efficient privacy-preserving federated learning and further propose an algorithm upon it for improving asynchronous federated learning of heterogeneous participants having non-IID data. We also propose an asynchronous stochastic gradient descent algorithm for general distributed learning of heterogeneous participants having non-IID data with convergence analysis. Experiments have shown the efficacy of our proposed approaches.

ACKNOWLEDGMENTS

I would like to express my greatest appreciation to my advisor Dr. Liqiang Wang, who has provided invaluable guidance and assistance to my doctoral research. He is a very kind, active and responsible advisor and is always available for recommending research directions with potentials, inspiring promising approach considerations, discussing possible problem solving techniques and providing continual support throughout the research process. My research would not be possible without his excellent mentoring. I also want to express my gratitude to other members of my graduate committee, Dr. Kien A. Hua, Dr. Yanjie Fu and Dr. Yunjun Xu, who have also provided very helpful suggestions to my doctoral research.

During my study and research, my parents have always provided me incredible and well-rounded supports. Their supports are most priceless to me and give me the confidence to persist in pursuing my goal. I will always cherish their encouragements and will try my utmost to strive for better.

My great thanks are also to my other colleagues, teammates, and anyone who has provided assistance to my graduate research. I admire them for contributing great cooperation efforts, holding timely discussions, providing insightful understandings and numerous suggestions to me whenever needed. Their assistance forms a significant component of my research and shows great teamwork spirit. My sincere thanks to all of them.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

One of the most significant phenomenon that accompanies the ever accelerating development of modern information era is the persistent generation of tremendous amount of data with both rapid generation speed and widely broad geographical distribution. Hidden inside such data are the abundant knowledge and information that intrinsically shapes the data generation. Such knowledge and information are vitally important for understanding substantial property of the data and could bring invaluable benefit to both the development of society and the improvement of people' daily life when adequately mined. This immense and still continually rising demand of obtaining the knowledge behind data stimulates the desire of many large-scale data analysis and learning tasks. Such tasks also present many special features and bring significant challenges to underlying hosting computing facilities and infrastructures. Proposing techniques and mechanisms that could tackle such challenges, fortifying the design of the computing infrastructure and fitting to the representative features of modern data analysis and learning tasks have important academic and practical significance.

Features of Modern Data Analysis and Data Learning Tasks

Based on their intrinsic characteristics, we can roughly categorize modern data analytical tasks into general data analysis tasks (for instance, those general tasks running on popular parallel and distributed platforms such as Apache Hadoop and Apache Spark) and data learning tasks (such as those tasks running with distributed learning, federated learning mechanisms). To better depict and understand modern data analysis and data learning tasks, we first inspect their most representative features. Such kind of data analytical tasks commonly possess features such as:

1. Data involved in the analytics can be large scale and can be geo-graphically distributed. This is due to the rapid and often widely spread generation of data, such that the data involved in an analytical task can be both with a large volume and with multiple origins possessing unique and irreplaceable data.

2. The amount of computing capability desired can be enormous. This is due to the frequently observed enormous amount of input data involved in modern big data analytics. The analytical workflows corresponding to such tasks often require processing all input data and corresponding intermediate results with multiple operations that could be on some extent paralleled, which favors larger amount of computing nodes within hosting facilities for efficient task completion.

3. Task can be time-critical. Modern data analytical tasks nowadays mostly on some degree have temporal urgency. This could simply be a general desire that the task be completed at earliest possibility, such that the obtained knowledge can be quicker applied to beneficial utilization. This could also be some more strict temporal deadlines that the analytical workflow should follow (thus be time-critical), which could also present in streaming type iterated data processing tasks.

4. Some of the data involved could be private. The large amount of data involved in analytical tasks are more than likely to contain some private data that are uniquely generated and owned by specific participants which shall not be shared. Protecting such private data and avoids privacy leakage at best effort could be an important consideration for the successful accomplishment of the according data analytical tasks.

5. Participants involved can have heterogeneous capabilities and non-IID data. Since data involved in an analytical task can be from multiple unique origins (participants), they can easily have non-IID data distribution. Furthermore, participants involved in computation

could also be from multiple agents in purpose of increasing available computing capability, this results in possible heterogeneous participants with different computing and networking capability.

6. It is possible to have a large amount of simultaneously submitted data analytic tasks in the hosting computing facility during a time interval. This is once again due to the rapid and spread generations of data that derive many diversified data analytical tasks which could happen concurrently. Thus from the perspective of computing facility, a computing platform may face task submission from multiple data analytical tasks simultaneously which could cause a resource competition condition.

Challenges Facing Computing Facilities

With the detailed description of certain features of modern data analysis and learning tasks, the focus now moves on to how the underlying computing facility could handle the load from all the data analytical tasks and provide efficient task completion. However, there is no denying that all the aforementioned task features bring much challenges to contemporary computing infrastructure and facilities. We summarize the desires and challenges to the computing facility as follows:

- Due to the extensive computing capability demand of large-scale data analytical tasks, facility's computing capability on one site (cluster) may not be sufficient for efficient task completion. And further scaling up on one site is likely to be more unrealistic due to cost, power and management concern. Therefore utilizing computing capabilities from multiple attending facilities becomes more actual. However, how to enable the collaborative mechanism remains a great challenge.

- From the view of the overall computing infrastructure which could be composed of multiple

distributed computing clusters, since there can be many continual and even multiple concurrent task submissions, how to reasonably schedule all analytical tasks to according computing cluster such that the overall efficiency is improved meanwhile lowering the missing deadline events of time-critical tasks is a challenge to the scheduling scheme.

- A special type of data analytical tasks is the data learning problem. Due to the features of the data origin, distributed data learning better fits the task requisites. Compare to synchronous organization form, asynchronous distributed data learning improves training efficiency by avoiding synchronization barriers. However, features of the underlying deployment scenario could bring more pressure to the asynchronous organization form. Such as, with heterogeneous participants having different capability and non-IID data, asynchronous methods need to take special consideration to grant a correct convergence result. How to provide efficient asynchronous approaches under specified deployment environment for efficient data learning remains a challenge.

- Since private data can be involved in the data analytical process, how to protect privacy during potentially geographically distributed analytical tasks is a significant requirement to the underlying system infrastructure design. Potential measures include keeping private data local with no cross-participant transmission and neutralizing private information contained in intermediate cross-participant information such as the participant gradients used in distributed learning. An overall effective privacy protection approach can help to stimulate participating enthusiasm of different data owners and alleviate privacy concern related to general collaborative distributed data analytical tasks.

Our Contributions

In view of previously mentioned task features and corresponding system challenges, my doctoral research focuses on developing a few approaches for the purpose of tackling aspects of these described challenges. And in hope that these works together could be towards more efficient collaborative distributed data analysis and learning.

- In order to enable large-scale data analysis of which the requirement of computing resources could be beyond the capability of one computing cluster, we propose a hierarchical framework that supports deploying a data analytical workflow onto multiple clusters with Apache Spark platform for collaborative data analytical task completion. This work has been published as paper [52].

- As stated previously, it is a challenge for accomplishing resource management in a large-scale computing system which may contain multiple distributed computing clusters. For scheduling a large amount of data analytical workflows potentially containing both general and time-critical jobs among multiple computing clusters, we proposes a reinforcement learning based resource management approach to improve overall efficiency and reduce violations of temporal deadlines. This work has been published as paper [51].

- We further proposes a deep reinforcement learning based elasticity-compatible resource management approach for heterogeneous computing environment containing multiple computing clusters. In this work, the nodes in different clusters may have different computing capabilities and certain clusters could have elasticity. This work has been published as paper [50].

- For collaborative distributed learning (including federated learning) among multiple participants as a specific type of distributed data analytics, we establish a new hybrid architecture

with the name HALE-Fed for efficient privacy-preserving federated learning. It could provide equivalent if not better participant-level privacy protection effect as the native Secure Multi-party Computation (SMC) meanwhile maintaining nearing to pure asynchronous organization training efficiency. We further proposes an algorithm Fed-SUDA which is built upon HALE-Fed for improving federated learning in a deploy environment with heterogeneous data and participants. It helps to enable a reliable asynchronous federated learning approach in such kind of deployment environments.

- We also proposes an algorithm named HP-ASGD as an asynchronous stochastic gradient descent (SGD) algorithm for general distributed learning having non-IID data and heterogeneous participants. It could correct the model shift concern of native asynchronous SGD method in such a heterogeneous condition and serve as a trustful SGD method in such a situation. We also establish the convergence analysis of the HP-ASGD algorithm for non-convex problems, so that the efficiency estimation and convergence guarantee can be quantitatively evaluated.

I hope the works done in this doctoral dissertation could serve as a step to stimulate more insightful and innovative approaches towards resolving challenges facing the distributed computing system from modern data analysis and learning tasks. And I hope this dissertation could contribute to the eventual goal towards more efficient collaborative distributed data analysis and learning.

CHAPTER 2: LITERATURE REVIEW

Parallel and Distributed Data Analysis

[1] Parallel and distributed data analytical systems have rising significance to meet the ever increasing demand from modern data analytical tasks. Many works have contributed into improving system performance in various aspects of such systems [28, 64, 56, 57]. Among them, Apache Hadoop and Apache Spark are popular and representative platforms for such data analytical systems which also attract attentions from researchers [34, 99, 76, 96, 97].

MapReduce is a popular computational model with great application and research potentials [40, 21, 12]. Apache Hadoop can be seen as a popular open-source implementation of the MapReduce paradigm. There are several existing research projects either providing a hierarchical level design for the MapReduce model or even providing framework design support for deploying MapReduce jobs to multiple cluster environments. [92] proposes a new Map-reduce-Merge model as an extension of the MapReduce paradigm. The new merge phase can merge heterogeneous dataset already partitioned and sorted by MapReduce and can express relational algebra operators as well as join algorithms. However, it increases system complexity and learning curve due to the introduction of several new components. [22] classifies MapReduce jobs into two categories based on whether they are recursively reducible or not. It provides a solution that could support hierarchical reduction or incremental reduction for recursively reducible jobs, however it is only applicable to single cluster environment. [30] introduces MRPGA (MapReduce for Parallel Genetic Algorithms), which additionally adds a second reduce phase to the original MapReduce model in order to address genetic algorithms, however, this extension is designed for a special application and may not be

[1] Content of this chapter is in part based on our published papers [52, 51, 50].

suitable as a solution for general MapReduce applications. The concept of "distributed MapReduce" is introduced in [9], which is a hierarchical design for MapReduce. However this solution is lacking scheduling algorithm and programming model design. [4] addresses the data analysis problem in the hybrid cluster, which consists of a local cluster and cloud computing resources, with the usage of both local and global reduce phases. However, the solution also lacks scheduling algorithm. Luo *et al.* [55],[54] presents a hierarchical MapReduce framework that adopts the Map-Reduce-GlobalReduce model introduced in the paper. The framework is capable of utilizing computational resources from multiple clusters to collaboratively accomplish MapReduce jobs, and it also provides scheduling algorithms for compute-intensive jobs and data-intensive jobs. However, above solutions are targeted for MapReduce paradigms, and are not designed for Apache Spark system.

Reinforcement Learning based Resource Management

Resource management for time-critical distributed computing faces many challenges, which requires a better understanding of both instantaneous and long-term influence of allocation decision. Traditional rule-based or white-box resource allocation models are inadequate on these goals due to intrinsic system complexity and difficulty in abstracting behavior characteristics. Instead, we tackle this problem by using the cutting-edge reinforcement learning technique, which is good at capturing intricate system features and gradually improving itself along RL process. Reinforcement learning is an important area in machine learning. The goal of which is to gradually learn to perform good actions in response to state representations of different environment status, in earning maximum action value.

Recently, many significant achievements have been accomplished using reinforcement learning technique. For one representative example, Deepmind [17] recently developed a computer Go program called AlphaGo to defeat world champions [73, 74]. In [73], they facilitate Monte-Carlo

Tree Search (MCTS) algorithm [11] with their own policy network and value network, which are obtained via techniques including supervised learning and reinforcement learning with deep neural networks. Further in [74], they apply reinforcement learning strategy directly without human knowledge, and are able to achieve model which surpasses previous one [73] in a short interval of time.

In the field of distributed computing, there are also studies focusing on utilizing various approaches in improving system performance and functionality. [47] proposes a search-based automatic parameter tuning method for MapReduce [16] framework. It uses a genetic algorithm to identify near-optimal configuration of several Hadoop platform parameters in minimizing job execution time. [86] proposes an architecture for scientific workflow management systems that supports provenance and atomicity to distributed scientific computations represented as scientific workflows. [52] proposes a framework facilitating execution of a big data computing application with multiple spark clusters. [79] proposes JDS-HNN, a heuristic approach that utilizes Hopfield Neural Network for job scheduling and data replication in a grid, in purpose of minimizing job execution makespan and data file delivery time. [94] and [83] try to deduce machine learning model that could respectively capture the relationship between execution time or percentage performance improvement with parameter configurations. The former is depicted as a regression problem, while the later a binary and multi-category classification problem. [1] proposes a cloud computing configuration optimization method for big data analytical platforms. It focuses on configurations such as selecting appropriate type and numbers of instances, and could distinguish a good configuration efficiently due to Bayesian optimization efficiency.

Approaches including reinforcement learning have been investigated to be applied to computing resource management. [2] introduces a parallel temporal-difference reinforcement learning algorithm for achieving optimal cloud resource scaling decision. [59] employs deep learning strategy in reinforcement learning to accomplish resource management in a cluster from gradually accumulated

experience and the result called DeepRM is comparable to state-of-the-art heuristics. [88] presents a novel multi-agent reinforcement learning method for load balancing problems of grid computing resources composed of multiple clusters with large-scale computing jobs. [41] proposes to enable model-free control in distributed stream data processing systems using deep reinforcement learning, which aims at minimizing tuple processing time in average. However, none of the above researches handles hybrid time-critical workload in distributed computing environment.

Elasticity-compatible Resource Management

Reinforcement learning (RL) has recently gained astonishing accomplishments in a diversified range of tasks such as artificial intelligence, object tracking, vehicle management, robot navigation and dialogue system [74][85][77][49][31][36]. Its advantages are well demonstrated in extracting knowledge from continual interactions with the environment even in complicated systems, which is otherwise hard to be abstracted by rule-based approaches even under expert guidance. Once trained, it could respond fast to inference for providing timely action decisions in contrast to searching-based approaches.

In accordance with this consensus, reinforcement learning has been adopted to deal with decision-making problems in a distributed computing environment. [59] presents "DeepRM" that utilizes reinforcement learning for obtaining a resource manager to coordinate workload execution in a cluster for improving execution efficiency. [88] uses multi-agent reinforcement learning in obtaining a method aiming at the job scheduling problem in a Grid computing environment, in purpose of realizing load balancing. In [20], a cloud controller towards resource allocation for applications in a cloud environment as an automatic workflow is obtained via reinforcement learning where techniques in accelerating training process are integrated. [2] applies Q-learning in training towards optimal policy for dynamic resource allocation for applications in a cloud. [48] tries to solve a joint

virtual machine resource allocation and power management problem by proposing a hierarchical framework learned via reinforcement learning. In [41], a model-free approach is obtained by deep reinforcement learning, targeting at minimizing average end-to-end tuple processing time for distributed stream data processing systems. [14] presents DRL-Cloud, a resource provisioning and task scheduling system achieved via deep reinforcement learning focusing on reducing energy cost.

[51] proposes a resource management approach for time-critical applications in a distributed computing environment via reinforcement learning. It is the most related one to our elasticity-compatible resource management approach. However, they differ in multiple aspects. Firstly, the underlying problem is largely different. Although similarly dealing with a multi-cluster environment, [51] does not consider cluster heterogeneity and elasticity, both of which are considered in this elasticity-compatible approach. Later descriptions in Chapter 5 will reveal how greatly this will change the problem nature and increase problem complexity. Secondly, depending on the new problem feature, the most important action value in reinforcement learning of Chapter 5 is vitally redefined with novel consideration and insights. Thirdly, the added problem complexity leads to a brand new model structure design. [51] uses a general neural network with fully-connected layers, while we in Chapter 5 propose a deep neural network based on LSTM structure and a partial network sharing multi-target learning mechanism. Fourthly, experiments in Chapter 5 provide thorough observations and show comparison with approach in [51]. Hence, this work shows great importance and significant difference comparing to [51] from aforementioned aspects.

Efficient Privacy-preserving Federated Learning

FL related: Federated learning [38] is a form of technique for organizing collaborative learning among multiple participants, meanwhile keeping their data local and private. It enables the opportunity to collaboratively learn knowledge and mine information from decentralized and privately

owned data of participants. FL is capable of handling up to a large scale of participants. In [6], it provides design principles for scaling FL to a large group of participants. [27] provides an application sample for FL at scale. The organization of FL can be traditionally classified into two cases, synchronous and asynchronous, with synchronous FL being the more popular one in the past due to its more formulated organization form. Besides SGD updating mechanism, other updating mechanisms such as FedAvg [60] which adopts multiple local updating rounds before coordinating to the global model, have also been developed for different algorithm balancing priority. However, synchronous FL could suffer from the synchronization overhead and asynchronous FL has drawn more and more attentions recently for its better efficiency due to the elimination of synchronization process. Thus there are works emerged that focused on the research of asynchronous FL. [13] proposes an asynchronous FL algorithm that tunes local computing episodes before sending out local updates towards accelerating training convergence. [53] presents an asynchronous federated learning mechanism for edge network computing which utilizes self-adaptive threshold gradient compression and dual-weights correction for network communication reduction.

Privacy related: [32] has provided a good summarization for recent advances and open problems in the field of FL. One category is about the Privacy related concerns related to FL. Similar as other form of machine learning algorithms, FL faces challenges from adversarial attacks. [72] shows membership inference attacks against machine learning models which is also applicable to FL. More correspondingly, [61] directly demonstrates the appliance of inference attacks against collaborative learning. [101] presents potential information leakage from gradients generated by model updating process, and shows that such leakage could be used for reconstructing information related to the training data. [68] extends the work in [101] by establishing the limits of the training data reconstruction in different network structures. As aforementioned, these attacks can be categorized into two functionality subgroups, one is those towards general information shared by many forms of machine learning, the other is more towards FL-specific participant level information. Regarding

security and defending measures, DP is an effective approach towards information obfuscation for defending multiple attacking methods and [24] provides description for applying DP in FL. Another popular security measure, especially in synchronous FL, is MPC/SMC technique, which hide individual gradient information by secure gradient aggregation among multiple participants and [7] establish a practical secure aggregation protocol that could be used in large-scale FL.

Hybrid FL related: There are studies related to hybrid-alike FL providing benefit to FL beyond either synchronous or asynchronous structure. [10] organizes asynchronous tiers for global updating with synchronous intra-tier computation with nodes of similar speed. [89] proposes a semi-asynchronous federated averaging protocol, including a lag-tolerant model distribution method that conditionally requires participant synchronization. [33] proposes a two-phase FL algorithm that selects a small model aggregation committee for training, where secure aggregation is only applied within committee to reduce communication overheads. However, none of the above work provide SMC-alike technique to a foundationally asynchronous FL framework, and none could solve practical issues like our proposed Fed-SUDA for asynchronous FL with heterogeneous environments.

Convergence Analysis of Distributed Learning Algorithms

Distributed learning (including FL) is a machine learning technique that aims at accomplishing collaborative learning from decentralized participant data [38, 93]. [6] shows the system design for large scale federated learning. [26] presents a distributed learning scenario among multiple agents.

Regarding convergence analysis related to distributed GD/SGD in FL: [35] and [87] present analysis for synchronous FL of convex function with non-IID data and full participation. [75] is with the similar setting as the above but for IID data. [84] and [100] then focus on FL convergence for

non-convex functions with full participation. [95] adopts similar setting but is with respect to the distributed momentum SGD. Accordingly for FedAvg [60] algorithm, [91] analyzes asynchronous FL with non-IID data and full participation. [43] shows result for synchronous FL of convex functions with non-IID data and partial participation. [70] changes setting in [43] to asynchronous but still for convex functions.

Other distributed learning with SGD has also drawn great attention from theoretical analysis. [69] presents parallelized SGD algorithm with lock-free manner and provides theoretical analysis for convex functions. [15] provides a martingale-based analysis that reveals convergence rates for convex functions with relaxed assumptions comparing to [69] and analyzes asynchronous SGD for non-convex matrix problems and for lower-precision arithmetic. [19] shows that asynchronous stochastic convex optimization achieves asymptotic optimal convergence rate. [46] further shows ergodic convergence rate for asynchronously paralleled stochastic gradient algorithms for non-convex functions in both network and multi-core systems. However, none of the above works is for asynchronous algorithm for non-convex problems with non-IID data and participant heterogeneity.

CHAPTER 3: COLLABORATIVE DATA ANALYTICS AMONG MULTIPLE CLUSTERS

[1] With the coming and on-going duration of the information era, more and more data are generated everyday, even in an exploding speed. These data carry lots of invaluable information that are of great importance to human society and global development. The necessity of analyzing such drastic amount of big data stimulates the continuing prosperous development of big data computing. Since the analytical process of such data are way over the computational capability of even the best single computing node, people spend great efforts to develop parallel computing methods and platforms. Among them, Apache Hadoop and Apache Spark are two of the most popular open-source big data computing platforms. Apache Hadoop utilizes HDFS (Hadoop Distributed File System) as its storage layer, and uses MapReduce computing model to provide end users a distributing computing platform that has better reliability and scalability than traditional parallel computing interfaces. Apache Spark further improves the performance of Apache Hadoop by introducing RDD (Resilient Distributed Dataset) object based on the in-memory technique. Apache spark overall provides better distributed computing performance for big data analytical workflow, which supports much richer computational operations and more complicated workflow structure comparing to Apache Hadoop.

Both Apache Hadoop and Apache Spark are deployed upon the concept of cluster. All computing nodes form a cluster that can be employed by the resource manager such as YARN to schedule computing tasks. In this case, all internal nodes are natively considered to have a local network connection with other nodes in the cluster. However, distributed computing may involve multiple geographical locations. Even if we use the virtual private network (VPN) technique to connect these computer into a single cluster, as the resource manager is not able to detect and realize such

[1] Content of this chapter is based on our published paper [52].

heterogeneous network structure, the cluster performance will degrade significantly. A hybrid cloud is a good example to utilize distributed computing on multiple geographical locations. A user may have one local cluster initially, but then realizes the shortage of computing resources due to data analytical demand, and decides to request more resources from public cloud platforms such as Amazon EC2. The user is then facing with the obstacle of how to integrate and utilize the resources from both local private cluster and public cloud computing recourses. On the other aspect, with ongoing popularity of cloud computing platforms, many organizations with data security concerns would like to keep sensitive data local. In this scenario, it is of great significance to enforce the isolation between multiple clusters, in order to obey the data security standard. For example, the data security standard may grant only the transferring of computation generated intermediate data but not original input data. These scenarios motivate great necessity of migrating big data computing workflow to multi-cluster environment.

In this chapter, we present a multi-cluster big data computing framework built upon Spark. Our major contributions include:

- A framework that addresses the problem of utilizing the computation capability provided by multiple Apache Spark clusters, where heterogeneous clusters are also permitted.
- A scheduling algorithm to optimize workflow execution on our multi-cluster big data computing framework.
- An integrated controller within the framework, which grants ability for submitting, monitoring, and finishing of workflows.

Architecture of Hierarchical Spark

The architecture of hierarchical Spark mainly contains two component layers, the global controller layer and the distributed layer. The global controller layer consists of the workflow scheduler and the global listener. The distributed layer consists of the distributed daemons, each has job manager and job monitor in charge of submitting and monitoring the job allocated to corresponding cluster.

When using hierarchical Spark, users provide the following files to the framework: (1) application files that contains component jobs to form the overall workflow; (2) configuration files that specifies application dependencies; (3) profiling information for component jobs. If the profiling information is absent, it can be supplemented by on-site profiling with the original workflow and sample data. Upon receiving input, the workflow scheduler extracts information, and then generates job allocation arrangement by our scheduling algorithm. Once the arrangement is decided, the workflow scheduler will utilize the global listener to distribute and start the actual execution. The global listener is another component of the global controller layer, based on the scheduling plan provided by the workflow scheduler, the global listener will deploy corresponding job to assigned clusters. It will also communicate with distributed daemons, which manage job submission and monitor job status. Once job finished, the distributed daemon will notify the global listener so that the latter will arrange transferring of the output file, submitting other dependent jobs, or launching the final job, until the entire workflow is finished. The architecture of hierarchical Spark is illustrated in Figure 3.1.

Workflow Model

In hierarchical Spark, the workflow model contains three type of components, *i.e.,* non-dependent job, dependent job, and final job. The non-dependent jobs are those jobs that start from initial input files, and have no other dependencies. The dependent jobs are those jobs that have dependencies on

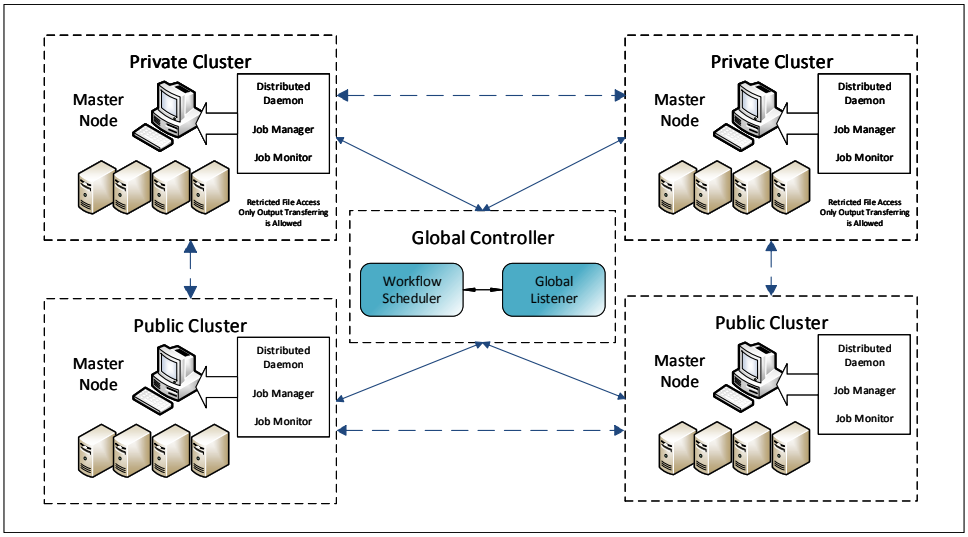


Figure 3.1: Architecture of Hierarchical Spark

other jobs, either non-dependent or dependent ones. The final job is the last component in the entire workflow, this is fixed to be executed on the central cluster. By dependencies, all these component jobs form the entire workflow as the input of our framework. Each job is a basic element that will be scheduled to clusters for computation. Figure 3.2 illustrates a hierarchical Spark workflow.

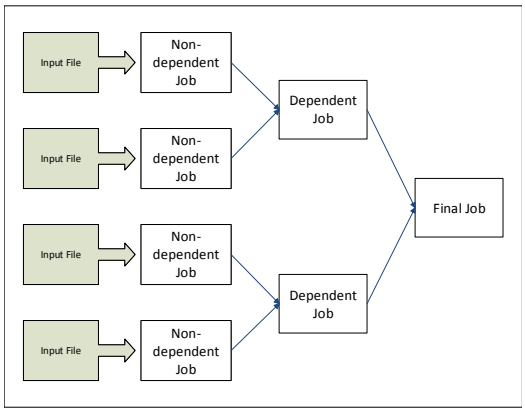


Figure 3.2: Illustration of Hierarchical Spark workflow

Recall in Spark, a job can be expressed as a DAG (Directed Acyclic Graph), similarly, our framework

workflow can also be represented as a DAG. The transformation from original spark workflow to the our framework workflow is natural.

Basically, Algorithm 1 starts by letting each stage in initial spark DAG become a job in the framework workflow. Then, we suggest splitting jobs in the workflow into multiple ones, or combine several jobs in the framework workflow into one job. We recursively examine the workflow until there is no new change. For example, in lines 3-8 of Algorithm 1, for a job in the workflow, if its “inputsize" over “blocksize" (equivalently the number of blocks) is very big, we suggest split it into multiple jobs, with default suggesting splitting number shown in algorithm. Conversely, in lines 11-18, by looking at some job i together with its dependency jobs list D_i as a group, we can check whether using the whole group as one job in our workflow is beneficial, then apply the change if this will significantly lower total transit data meanwhile not violating other constraints. We provide general default values to thresholds in the algorithm, for example, the default value of threshold1 is $10\times$(max # of executors in all clusters). However they can also be specified by the user to customize the suggestion engine.

Using the wordcount application as an example. When it is used as the workflow input for our framework, Algorithm 1 will provide transformation plan from this Spark workflow to our framework workflow. For this example, if the number of block for the input file is larger than threshold1, we suggest split the job into multiple ones. In this case, the split can be accomplished by roughly repeating the unary operation “reducebykey()" twice, with each new non-dependent job taking care of one portion of the initial input file.

The core pesudo code for original wordcount application is follows.

$$line.split(\text{“ ”}).map(word \rightarrow (word, 1)).reducebykey()$$

Core pesudo code for our framework wordcount application (non-dependent jobs) is the same with

the code above, which also demonstrates that the transformation burden is little to framework users. The core pseudo code for the final job is:

$$collectedresultline.map(parser(\text{“}K,V\text{”} \rightarrow (K,V)))\ .reducebykey()$$

Algorithm 1 Spark Workflow Transformation Algorithm

```
1:  Let each stage in Spark become a job
2:  for each job i do
3:      if (D_i = ∅ && (β =# of cluster_hasinput)>1 && input/blocksize > threshold1) then
4:          split job to β jobs, update corresponding dependency lists
5:      end if
6:      if (D_i ≠ ∅ && input/blocksize > threshold1) then
7:          split job to input/(blocksize×threshold1) jobs, update corresponding dependency lists
8:      end if
9:  end for
10: Change=true
11: while Change==true do
12:     Change=false
13:     for each job i do
14:         if !IsNewSplittedJob(i) && total input blocks to i <threshold1 && new output/original
            total output<threshold2 && combined input exists if non-dependent jobs are involved
            then
15:             combine job i and D_i into one job, update corresponding dependency lists; Change=true;
16:         end if
17:     end for
18: end while
```

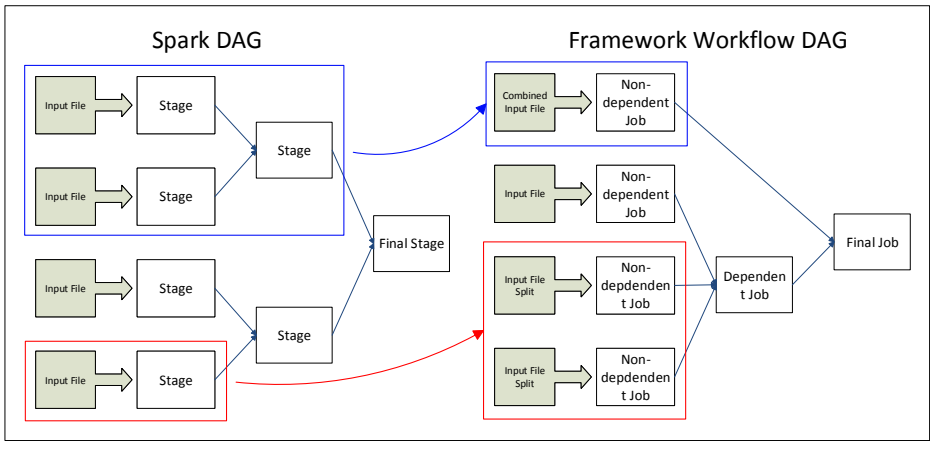


Figure 3.3: Illustrative framework workflow DAG generation

This final job is added to act as an eventual collection and reduce procedure for all intermediate data generated by previous non-dependent jobs, its code is straightforward and easy for framework

users to add.

Figure 3.3 illustrates a more general case, by showing the original spark DAG on the left and Framework workflow DAG on the right. It shows that the general suggestion result from our algorithm which may include some splitting as well as combing, with other stages in the original DAG directly becoming corresponding jobs in our new workflow.

Scheduling Algorithm

Our framework not only aims at enabling distributing component jobs of an entire workflow to multiple spark clusters for cooperated computing, but is also equipped with scheduling algorithm designed to better achieve multi-job & multi-cluster scheduling in purpose for better performance. Our proposed algorithm is shown in Algorithm 2.

Multi-job & multi-cluster job scheduling is a well-known NP-hard problem. To achieve a good solution in an efficient way, we use simulated annealing as the major heuristic algorithm for solution searching. To further increase the efficiency, we use greedy algorithm to achieve a better initial solution that will be provided as input to the simulated annealing algorithm.

Regardless of the choice of heuristic algorithms, the core design of our scheduling algorithm is nonetheless the evaluation function that could assess the scheduling arrangement. Since our aim is to reduce the total execution time of the entire workflow, our evaluation function is designed to be capable of evaluating the running time cost of a specific scheduling arrangement. Further, the evaluation function needs a performance model, which can provide us an estimation for the running time of a job on a cluster.

Performance Model

Now, we provide the performance model that could estimate the running time of a job on a candidate cluster. To better introduce the entire performance function, we first introduce the performance model for a stage in a job. Its details are as follows:

Let l be the average computing time of a task in a stage using its required executor. We define it in unit of second. Let c denote the total available executors in a cluster for this job. Let a denote the number of tasks in a RDD in a stage, we choose the maximum number of tasks in a RDD in a stage if RDDs in a stage have different number of tasks.

Thus, a/c represents possible waves during execution.

The performance model for a stage is defined as:

$$t_{stage} = l \cdot a/c \tag{3.1}$$

Now, we propose the performance function that models the execution of a job into a more detailed level as running of stages, relevant to the stage running concept in Spark.

$$PF = \frac{InputSize}{SampleSize} \times \left(\sum_{stages} t_{stage} + \sum_{shuffles} ST * nF \right) \tag{3.2}$$

where ST is the intermediate data shuffle time that happens between stages. If no shuffle exists between some stages, corresponding shuffle time equals zero. nF is the network factor, which can reflect the different internal network speed of cluster. Notice that we consider the possibility that the profiling may be corresponding to a sampling data, instead of the entire input data, so the ratio

of $InputSize$ over $SampleSize$ is also considered in the formula.

This performance function is currently adopted in our framework. Nonetheless, we would like to point out that, when applicable, depending on different coarseness of profiling data, the performance function can certainly be replaced by even more specially designed or more complicated performance models suitable for certain scenarios corresponding to the actual application.

Scheduling Algorithm and Evaluation Function

When designing the scheduling algorithm for our framework, we have considered different options at the early stage. The non-dependent jobs in our framework are a little special, as each of them has no dependencies, thus all can be submitted at the beginning of the execution. Consider these jobs as a group, one option for scheduling is the initial optimizing scheme designed to focus on optimizing the arrangements of this group for better performance. As stated before, to decide an optimized plan for all jobs in this group, we use greedy algorithm (sort all stages with computing load in descending order) to achieve an arrangement plan, and then an on-the-fly arrangement plan for other jobs in the workflow in actual submission time order. In this scheme, the arrangement plan for all non-dependent jobs will not consider the consequences it brought to other dependent jobs which depends on them, it therefore can be seen as a non-forward-looking scheme.

Stimulated from this idea, but further improved, we design a global optimizing scheme aims at providing an overall scheduling arrangement of the entire workflow. We want to take into consideration the consequential effect of optimizing non-dependent jobs it may bring to the later jobs. In other words, instead of providing current moment optimized arrangement plan, we would like to take global vision, foresee the whole picture of workflow DAG, then decide an arrangement plan for each job in the workflow. Due to its more advanced feature, we select this scheme as our framework scheduling scheme. The entire arrangement plan will be decided based on the following

procedure:

Firstly, similar as proposed in the initial optimizing scheme, we obtain arrangement plan for all non-dependent jobs by greedy algorithm. Now, for each non-dependent job, the scheduling plan for it will be represented as $(cluster, t_{start}, t_{finish})$, where $cluster$ is the selected cluster, t_{start} is the job start time. t_{finish} is the job finish time generated by the performance function.

For all dependent jobs, the tuple $(cluster, t_{start}, t_{finish})$ can be filled in by first calculating job_{st} using equation 3.3:

$$job_{st}[s'][j] = \max(t', cluster_{at}[j]) \tag{3.3}$$

where $t' = \max_{s}\{t_{finish}\ of\ s + IMO(s,s') | s \in dep\ set\ of\ s'\}$, $IMO()$ is the intermediate output transferring time for two jobs, $cluster_{at}[j]$ is the available time of cluster j which s' may depoly to. Equation 3.3 is also used for non-dependent jobs by considering their dependency set as empty and only use clusters that has its input.

For any job where its dependencies has been cleared, its $job_{st}[s'][]$ times are available. All these jobs can gradually be fit into the greedy algorithm. Once their cluster arrangements are decided, the corresponding t_{start}, and t_{finish} components can be filled in. Recursively, this can form an entire initial scheduling plan for the entire workflow, this part is shown in lines 3-33 in our proposed Algorithm 2. In fact, during this process, we are already simulating the execution process of the workflow together with the usage of greedy algorithms for achieving an initial scheduling plan. The similar idea of simulation will be used as the evaluation function in the iteration process of the simulated annealing algorithm, where t_{finish} of the final job is the eventual output of the evaluation function.

Secondly, for simulated annealing, each time it will randomly change one cluster arrangement if suitable, feed into the evaluation function $E()$ to simulate its execution to achieve the new t_{finish} of

the final job, which is the result of the evaluation function $E()$. Based on its result and therefore the result of the function $P()$, the simulated annealing can decide whether to keep the current plan or to update to the new one. The idea of the simulated annealing is that, if the new arrangement is better than the current one, it will always accepts it; however, if the new arrangement is worse than the current one, it will stochastically accepts it, which helps in jumping out of the “local" extrema. The probability depends on the parameter αT and the evaluation difference. At the beginning of simulated annealing, the result of $exp()$ function if used is very close to 1 and therefore there is much higher chance for accepting bad arrangements. In contrary, when nearing to the end of the algorithm, T, therefore αT, becomes small and the chance of accepting bad arrangement is significantly decreased. This simulated the physical annealing process where T act as the “temperature" in original process. This algorithm is good at obtaining global extreme for scheduling arrangement. Parameter α is used to make sure that according to the range of evaluation function difference, the initial probability to accept bad arrangement is very close to 1. The function $P()$ used in our simulatedAnnealing() function is:

$$P(E(S),E(S_{new}),T) = \begin{cases} 1 & IF\ E(s_{new}) \leq E(s) \\ \exp\left(\frac{E(s)-E(s_{new})}{\alpha T}\right) & Otherwise \end{cases} \tag{3.4}$$

The SimulatedAnnealing() function is shown in lines 35-43 in Algorithm 2. To supplement the detail, we now will state the definition of the evaluation function $E()$, which is a simulation process, as follows:

The input for the simulation engine is the set of jobs, each with a configuration tuple $(cluster, t_{start})$. The simulation process will then generate a simulated running for each cluster. For each cluster, the input is the tuples with format (job, t_{start}). The simulated cluster submit the job to it by order of start

time. For all jobs with t_{start} time undecided, the cluster simulation will wait until its dependencies are all cleared. Then each job on the cluster simulation will simulate its running by result from the performance function, with currently updated cluster information being considered. Simulations for all clusters are processed simultaneously. Eventually, all simulation of all clusters are finished, then the t_{finish} of the final job becomes the eventual output of the evaluation function. An illustration graph of this process can be found in Figure 3.4. Now, we can formally state our scheduling algorithm in Algorithm 2.

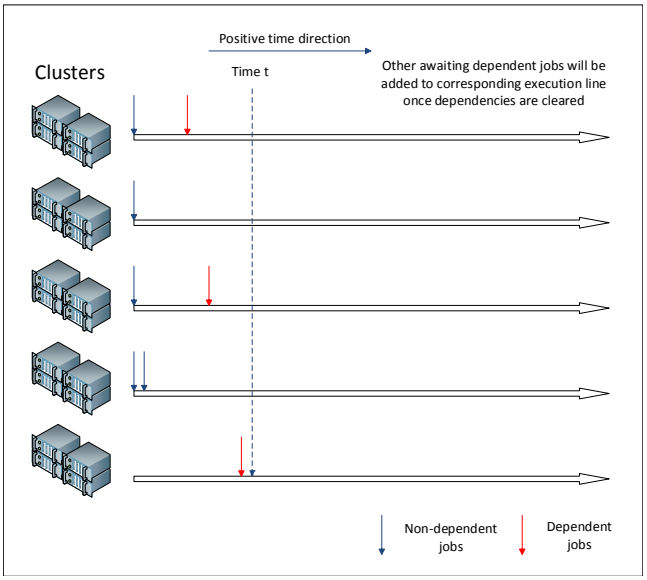


Figure 3.4: Cluster running simulation

Implementation Issues

Global Controller and Distributed Daemon

To enable the distributing, monitoring and executing of the big data analytical workflow, we designed global controller and distributed daemon in our proposed multi-cluster big data computing

framework.

The global controller is composed by the workflow scheduler and global listener, the workflow scheduler accepts user provided framework input, extract necessary information, and provide the scheduling arrangement result. The global listener is constructed as a multi-threading program, where all distributed daemon will be connected to it. Functionality of the global listener include: Send job arrangement to corresponding distributed daemon, order the moving of intermediate files.

The distributed daemon are composed by the job manager and the job monitor. Upon receiving job arrangement, the distributed daemon will submit the job to the cluster, it will also capture the application ID after submission, using it to monitor the job status. Once succeed, the distributed daemon will notify the global listener about the status update and intermediate file collection movements will be applied if necessary.

File Transfer

During the execution of the workflow, there are certain steps that contain or require intermediate data file transfer. For example, the final job definitely relies on outputs generated by some other jobs in the workflow. Also, when deploying dependent jobs, it may be necessary to apply movement of intermediate data to other clusters. In our framework, we assume the underlying file system to be HDFS (Hadoop Distributed File System). In order to make the file transfer more efficient, instead of applying the HDFS to local, transfer, then local to HDFS procedure, we use transfer command provided by the HDFS API (hdfs distcp) to facilitate direct and efficient parallel file transfer between source and target HDFS system.

Algorithm 2 Scheduling Algorithm

Create $Scheduling[]$, cluster available time $cluster_{at}[]$. For each job i, create estimated job start time $job_{st}[i][]$, dependency list $D[i][]$, and the wall clock time for all jobs in dependency lists of job i, *i.e.*, $Dep\text{-}walltime[i][]$. For each cluster j, create $Running[j][]$ to record running interval and capability usage of each job on cluster

GreedySolution() {
while RemainingJob!=0 **do**
  for each job i where $Scheduling[i]$=Null **do**
    if $D[i][]$==Null **then**
      scheduling-pool.add(job i)
      $job_{st}[i][]$= Eqn(3) result by $Dep\text{-}walltime[i][]$
    end if
  end for
  if All jobs in pool has same min start time **then**
    Sort(scheduling-pool, computing load, descending)
  else
    Sort(scheduling-pool, $\min_j(job_{st}[i][j])$, descending)
  end if
  for each job i in scheduling-pool **do**
    initialize $Score[]$;
    for each candidate cluster j **do**
      // Using reciprocal of estimated job finishing time as score for cluster j w.r.t job i for scheduling
      decide cluster j capability by $Running[j][]$ and $job_{st}[i][j]$
      $Score[j]$=$1/(job_{st}[i][j]+PF(i,j))$
    end for
    j=$Scheduling[i]$=$argmax(Score[])$
    RemainingJob$--$
    $job_{walltime} = job_{st}[i][j]+PF(i,j)$
    update $Running[j][]$
    if cluster j is fully occupied by submitting job i **then**
      $cluster_{at}[j] = \min(job_{walltime})$ for current jobs on j
    end if
    Delete job i from all dependency lists, adding $job_{walltime}$ to corresponding $Dep\text{-}walltime[][]$
  end for
end while
return Scheduling }

SimulatedAnnealing() {
for k = 0 through kmax **do**
  $T = 100 \times (\sqrt[kmax]{0.001})^k$
  Scheduling' = randomAlternation(Scheduling)
  if P(E(Scheduling), E(Scheduling'), T) >= rand(0, 1)) **then**
    Scheduling=Scheduling'
  end if
end for
return Scheduling }

Experiments

Our experiments are done on Amazon EC2 cloud computing platform, the Hadoop version is 2.7.3, and the Spark version is 2.1.0. Each cluster used in the experiment utilizes at most 9 m4.xlarge computing nodes to compose cluster of different sizes. For each cluster we mention below, the

number of nodes are referring to data nodes (computing nodes) in the cluster and there will be an extra name node in the cluster as well. We set one executor on each computing node that utilizes four virtual cores. There are mainly two purposes of our experiments. First, to show that by enabling multi-cluster collaborative execution, we could dispatch the original workflow by component jobs that could run on different clusters. Second, in some situations, the distributed workflow can also outperform the original application due to enabling of computing resources from multiple clusters and our designed scheduling algorithm. Our first experiment uses the WordCount application.

- WordCount: Comparing Effort of Original and Distributed workflow

In the first experiment, we run WordCount on 100GB input file with a 6-node cluster, this will act as our execution for the original workflow and as the comparing case for other distributed workflows.

For comparison, we run the distributed workflow on two, three, and four clusters (one of them is the central cluster where the final job is on), each having 6 computing nodes, and each deals with its proportional portion of the original total input. We further assume all clusters have all inputs in this experiment. The distributed component jobs are the same as the original WordCount job, however, in the end, the output files will be collected to the main 6 nodes cluster, and apply an additional application which act as a global reduce process. For 100 GB input on original workflow on one single cluster with 6 computing nodes, the total execution time is 16 minutes. As an example, in comparison, for three cluster scenarios, the 33 GB input on 6-node cluster costs a maximum of 5.6 min execution time, the generated output file is around 2 MB for each cluster, gathering them to the main cluster using HDFS distcp will cost around 20s, and the final job running time on the 6-node main cluster will cost about 49 seconds. Due to the scale of the workflow, other overhead caused by the architecture is low enough to be ignored, in fact, the scheduling algorithm can even be omitted in this special case. Therefore, the total performance comparison is 16 minutes vs 6.8 minutes, which yields a 2.35 times speed up. The result related to all number of clusters is shown in

Figure 3.5.

- WordCount: Comparing Effect of Default and Our Proposed Scheduling Algorithm

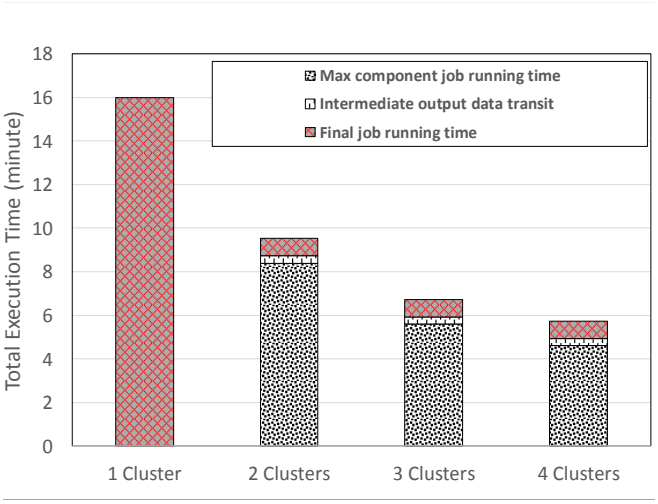


Figure 3.5: Wordcount workflow execution time comparison

In the second experiment case for WordCount, the scenario simulates where there are four clusters, with 2, 4, 6, 8 computing nodes respectively. The 8 nodes cluster is the central cluster where the final job is on. Four identical component jobs are split from the original WordCount computing, each deal with 1/10, 1/5, 3/10, 2/5 portion of the 100G input, proportional to the component cluster's computing node numbers. The entire computation process is the same as in the first experiment. We further assume all clusters have all inputs in this experiment. Now, suppose we adopt the default fair scheduling algorithm in Spark to our framework. If all workloads are in descending order of their input size (as well as computing burden in this case), but all cluster are in ascending order of their number of nodes, then by fair scheduling, the heaviest task will be arranged to the smallest cluster, etc. However, for our scheduling algorithm, no matter what the sequence of the workloads and clusters are, the scheduling will make the correct decision to send corresponding component

jobs to the cluster that is proportional to its computing load. We now show the experiment result in Table 3.1.

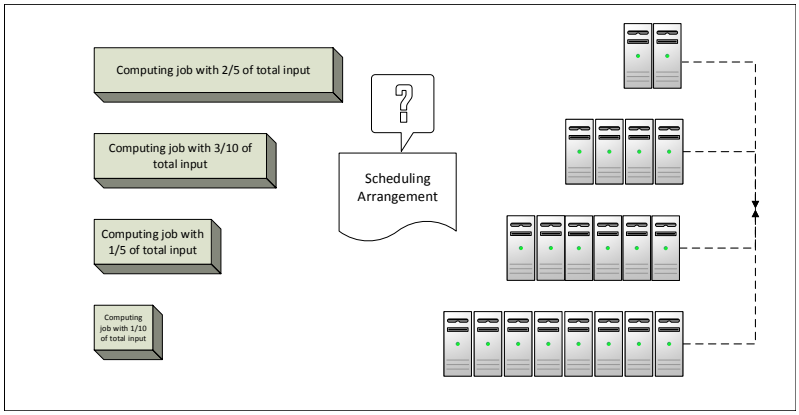


Figure 3.6: Illustration graph for component jobs and clusters

Table 3.1: Component finishing time with different scheduling schemes

	Fair Scheduling		Proposed Scheduling	
Cluster	Input	Finish Time	Input	Finish Time
cluster-1 (2 nodes)	2/5	19 mins	1/10	5.0 mins
cluster-2 (4 nodes)	3/10	5.2 mins	1/5	5.2 mins
cluster-3 (6 nodes)	1/5	5.2 mins	3/10	5.2 mins
cluster-4 (8 nodes)	1/10	5.3 mins	2/5	5.3 mins

We can observe from the result that, for the default fair scheduling scheme, the longest component job running time is 19 minutes, whereas for our proposed scheduling algorithm, the longest component time is 5.3 minutes, since all intermediate outputs from all components jobs are all very similar in sizes (about 2 MB), and the running time for the final process job will be very similar as well, the running time improvement in the component jobs will greatly be reflected in the overall execution time. Therefore, our proposed scheduling algorithm is better than the default scheduling scheme in Spark. In fact, for component jobs only, the maximum running time achieves a 3.58

times speedup by scheduling arrangement improvement.

- GIS Analytical Workflow: A Practical Workflow Demonstration on Hierarchical Spark

In [98] and [76], some GIS (Geographical Information System) computations have been accomplished on parallel computing platforms, especially in [76], these computations are executed on Apache Spark platform. Such computations include geographic mean computation, geographic median computation, etc. Stimulated by such application, in this experiment, we try to distribute an actual GIS workflow to multiple clusters by our framework. The workflow uses users' twitter sending GPS positions with format (userID, lon, lat) as input, accumulated by user ID, calculate their geographic mean and median, then join the results by user ID again to achieve a tuple for each user that could describe the geographical social behavior center for the user for further analysis, the output format is (userID, Geographic Mean, Geographic Median). The definition of geographic mean and median, together with a workflow illustration graph in Figure 3.7 is shown below:

Geographic Mean:

$$LON = \frac{\sum_{i=1}^{n} lon_i}{n} \qquad LAT = \frac{\sum_{i=1}^{n} lat_i}{n} \tag{3.5}$$

Geographic Median:

$$Median = \min_{x \in space} \sum_{i=1}^{n} \sqrt{(x_{lat} - lat_i)^2 + (x_{lon} - lon_i)^2} \tag{3.6}$$

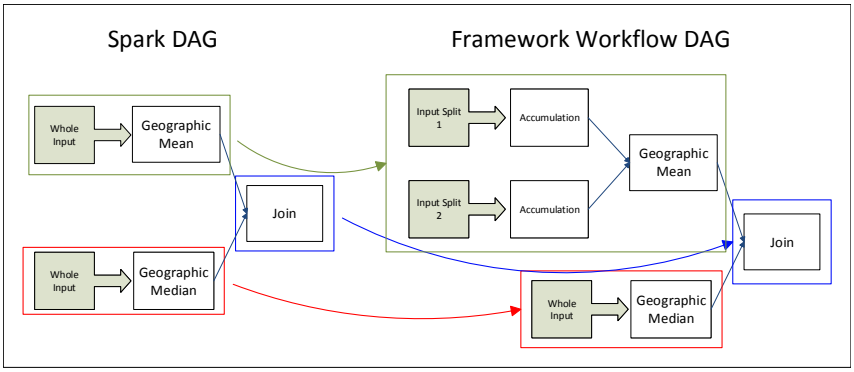


Figure 3.7: Illustration graph for Spark DAG workflow and framework DAG workflow

This experiment shows a scenario with three component clusters. The first cluster has 4 nodes and is a private cluster with half of the whole input data, which are sensitive. The second cluster has 4 nodes and is a public cluster with second half of non-sensitive data. The third cluster has 8 nodes (central cluster) and is a private cluster with whole input. In the framework workflow, since the geographic median computation requires whole input, it is kept as one job and deployed to third cluster. The original geographic mean computation is firstly split into two accumulation component jobs, with each one takes half of whole input on the first and second cluster respectively, accumulates their GPS locations and number of occurrence, generate output in format (userID, accumulated lon, accumulated lat, number of occurrence). Then, the geographic mean job in our framework workflow which reduces the intermediate outputs from two accumulation jobs is launched on the first cluster. Eventually, both intermediate outputs from geographic mean and geographic median jobs are gathered to central cluster, where the joining of the two intermediate results by userID is launched to achieve final result desired.

The execution time comparison is shown in Table 3.2. It shows that our framework not only enables the collaborative execution of this workflow on multiple clusters, but also achieves performance

improvement comparing to the original single cluster execution. It is also worth mentioning that this experiment well demonstrates the capability of our framework in maintaining certain data security and isolation standards.

Table 3.2: Component job finishing time and total execution time of framework workflow (In comparison, total execution time in one cluster is 3.3 min)

Framework workflow	Time
Accumulation 1	55 s
Accumulation 2	59 s
Geographic Mean	23 s
Geographic Median	1.5 min
Final Join	45 s
Total Execution	2.8 min

Summary

In this chapter, we present our proposed hierarchical Spark framework. The experiments show that the proposed framework not only enables the functionality to distribute original spark workflow to multiple clusters for collaborative execution, but also provides great performance improvement due to better utilization of the overall computing resources.

CHAPTER 4: RESOURCE MANAGEMENT FOR TIME-CRITICAL COMPUTING IN A MULTI-CLUSTER ENVIRONMENT

[1] In today's big data era, enormous amount of data is generated continuously and awaiting to be analyzed. Some of the analytical applications, such as accumulative historical data statistics analysis, may not have strict deadline or are more tolerant of response delay. However, more and more data analyzing applications, such as streaming data processing, highly rely on timely response from execution result, and can be referred as time-critical jobs. In this work, we classify time-critical applications into two subcategories, time-critical streaming application with approximately periodical repeating patterns and non-streaming single running time-critical application without repeating patterns.

Due to frequent appearing of gigantic amount of data and deeper analytical workflow in contemporary data analytics, these applications often rely on large scale distributed computing systems which often include multi-cluster/multi-group topological structures due to geographical distribution of resources, internal isolation for resource management purpose, possible hybrid cloud elastic structure of computing capabilities, etc. In real-world environment, it is more likely that the large scale computing resources not only handle time-critical applications but also regular non-time-critical data analytics as well. We call such kind of mixed workloads with time-critical and non-time-critical applications as *hybrid workloads*. Although resource scheduling and management have been well studied in traditional parallel and distributed computing, time-critical big data analytics brings many new challenges such as how to balance multiple performance demands regarding hybrid workloads, how to at best effort fulfill the temporal needs from time-critical applications meanwhile keeping general job delay low.

[1] Content of this chapter is based on our published paper [51].

In this chapter, we present a reinforcement learning based resource management approach that takes into consideration many unique features specific to efficient resource utilization on large-scale distributed data analytical systems for hybrid workloads. Our approach coordinates overall cluster-level resource allocation and is compatible with different inner-cluster resource management components. By our approach, time-critical and non-time-critical applications can comprehensively utilize computing resources in an efficient and harmonic way. The experiment results demonstrate that our approach is more effective than rule-based resource management approaches due to better capability of capturing complex characteristics behind scene using reinforcement learning. The major contributions of this chapter include:

- Use reinforcement learning based approach with proper neural network to obtain effective resource management solution that outperforms baseline rule-based resource manager.
- Design applicable value function definition in reinforcement learning for evaluating both effects of actions in reducing missing deadline occurrences and reducing average job delay.
- Improve reinforcement learning technique relating to ε-greedy strategy, as well as other accommodations to better suit RL approach to underlying practical problem and improve learning effectiveness and efficiency.
- Compare effects of different reinforcement learning models with multiple competitive rule-based resource management schemes in various hybrid workload scenarios.

Problem Description

The overall computing resource is defined as multiple computer clusters with specific capability. To define computing capability, we use the concept of executor in Yarn [82], which is a stationary basic allocation unit of a combination of virtual CPUs (*a.k.a.,* vCPUs) and memory. This concept is

general and is widely adopted in popular distributed computing systems such as Apache Hadoop and Spark. Clusters can be heterogeneous in the sense that they provide overall different numbers of executor units. In this work, we assume executors in a cluster are identical to different applications so that they provide the same computing performance.

The workload deployed to the overall computing resource is hybrid. More specifically, it can be a combination of three categorical applications: (1) streaming applications that repeat executions of themselves with different batches of arriving streaming data, which intrinsically contain temporal desire thus are time-critical; (2) non-streaming time-critical applications without repeating patterns; and (3) regular non-time-critical applications. Each application may have different duration, execution time on single executor and different computing capacity needs in unit of executors. Streaming applications can also possibly experience streaming input data fluctuation. In such a case, the streaming application will reflect this as a different desire for numbers of executors in order to maintain equivalent workload on single executor to better achieve temporal requirement in data fluctuation. In addition, we assume that computing resources are released between intervals of batch executions of streaming applications for less idling resource occupation.

At each moment, multiple applications may possibly arrive at the scheduling pool, which mimics the practical scenario where multiple users submit jobs simultaneously. At each moment, the resource management module picks one of the awaiting applications to deploy onto one of the computing clusters for actual execution. Selecting which application and which cluster is crucial in determining the overall performance of all applications.

Internal scheduling within a cluster is accomplished by local scheduler. The global resource management module does not interact or intervene with the operation of the local internal schedulers. As a matter of fact, this design brings up one advantage of the global resource management module, that is, our proposed reinforcement learning is capable of coordinating with different local internal

scheduling schemes, which greatly enhances the generality and applicability of the proposed approach to different practical scenarios.

For time-critical applications, if the actual execution time exceeds its temporal deadline requirement, this triggers the “missing deadline" (MDL) event. It can be caused by prolonged resource allocation waiting time and/or insufficient executor allocation due to resource competition among concurrent applications. For both time-critical and non-time-critical applications, we also generally care about the average job execution delay ratio, where a lower average of job execution delay ratio represents more efficient overall application running.

A critical problem that the global resource manager needs to solve is how to schedule a hybrid workload for minimal total missing deadline events and average job delay ratio. Hence it may need to abstract cluster running statistics, application characteristics, workload pattern, and internal local scheduler behavior. The achieved resource management module should be capable of achieving favorable balance in goals and provide effective resource management scheme.

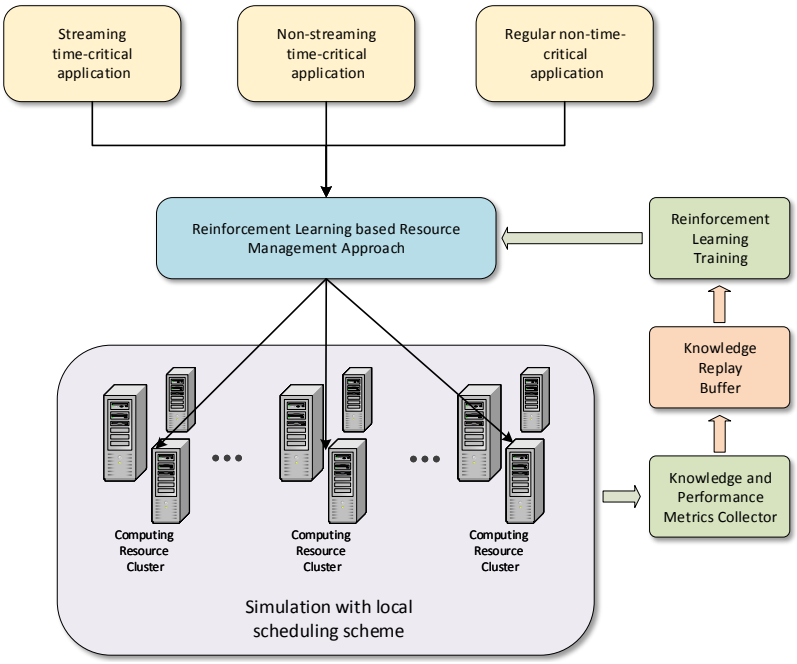


Figure 4.1: An illustration of the architecture of our approach.

The Reinforcement Learning Based Approach

As depicted in Figure 4.1, we propose a resource management approach based on reinforcement learning for hybrid workloads in distributed computing environment. Recall the main purpose is to select an application from possible multiple candidates and identify a suitable cluster for it among all available ones for application deployment.

Concept Definition and Value Function Design

The *environment* of our reinforcement learning can be described as follows: the overall distributed computing resource is composed of multiple computing clusters. These clusters may provide

different numbers of executors, with each executor a fixed basic combinational allocation unit of vCPU and memory resources. The applications awaiting to be allocated can be categorized as three classes: streaming time-critical, non-streaming time-critical, and regular non-time-critical.

The *state* of the environment is defined as a vector of several components, with details shown in Table 4.1 according to experiment setup. (1) For each cluster, a component is used for describing its computing capability in terms of executors, discrete resource utilization statistics in the past 100 time slices if available, and total occurrences of missing deadlines for time-critical applications in current episode. (2) Another component is for job profile. Without loss of generality, we use streaming time-critical application as an example for description. The profile includes: (a) The category of job: streaming time-critical application, non-streaming time-critical application, or non-time-critical application; (b) Discrete probability distribution of resource utilization due to streaming data fluctuation; (c) Single executor occupation time, execution deadline for batch running, and overall duration of the streaming application. Applications in the other two categories can nonetheless use the same state format with intuitive modifications. For all state representations, we uniformly expand or shrink individual value range accordingly to maintain similar data range for different dimensions in state vector.

Table 4.1: State representation in reinforcement learning model

Vector Component	**Dimensions**
Cluster ($i = 1 \ldots 5$)	
Sequence number	1
Capability	1
Occupation statistics (latest 100 steps)	100
Current total missing deadline	1
	515 ($\mathbf{103} \times \mathbf{5}$)
Application	
Category	1
Discrete resource utilization distribution	10
Single executor occupation time	1
Execution deadline	1
Duration	1
	14
Overall state vector	**529** ($\mathbf{515} + \mathbf{14}$)

The *action* space of our reinforcement learning model is composed of a number of actions equaling the number of candidate clusters from 1 to C. In our experiment, $C = 5$.

An *episode* is defined as the successful finishing of a fixed number of applications, where each application finishes its execution (for non-streaming applications) or a number of repeating executions (for streaming applications) during its lifespan.

The *value* of an action with respect to assigning a time-critical application i to a cluster can be defined as follows:

$$value_i = -\lambda \cdot MDL_i - \gamma \cdot MDL_{t>t_i} - \eta \cdot DelayResult_i \tag{4.1}$$

Similarly, the *value* of an action with respect to assigning a regular non-time-critical application i to a cluster can be defined as follows:

$$value_i = -\gamma \cdot MDL_{t>t_i} - \eta \cdot DelayResult_i \tag{4.2}$$

where MDL_i is the accumulative occurrence of missing temporal deadline events of application i during its lifespan. $MDL_{t>t_i}$ is total number of missing deadlines for all applications in entire resource that happen after t_i when application i is deployed and before eventual termination of i. The delay result $DelayResult_i$ is the average value of all running time ratio of application i in possible multiple executions during the episode. A running time ratio of application i is the ratio of its actual running time over its optimal expected running time. λ, γ and η are hyper-parameters. In our experiment $\lambda = 1$, $\gamma = 0.02$ and $\eta = 0.1$.

The *value* of an action here is equivalent as accumulated rewards an action received in an episode. Definition of the *value* function can be interpreted as follows. Our purpose for resource management for hybrid workloads is to increase the overall resource utilization of all clusters by assigning newly coming applications, meanwhile achieving multiple performance considerations. For each application, we would like the average job delay result remains low. However, for any time-critical application, we also want to greatly restrict any temporal deadline missing. Consequently, these should be comprehensively taken into consideration by desired resource manager. Specifically, our value function not only depicts the influence of selected action to the occurrence of application's own missing deadline events, but also takes into account the influence of it to all missing deadline events happened after the action, regardless of whether the event is solely related to application i or not. This ensures the global vision of action value evaluation, and enables consideration of correlated influence of multiple actions during action selection performed by the approach.

Neural Network and Reinforcement Learning Method Design

To capture important characteristics of efficient resource management decision, we utilize a neural network of three layers, including two fully connected hidden layers of 2000 and 500 neurons respectively and one output layer with neurons equal to the number of available actions.

We use reinforcement learning framework to improve the neural network model served as a value function estimator. Given feature vector input regarding to a single application, the network outputs value estimation for each possible actions. Action is selected based on ε-greedy strategy.

To accomplish the training process, in each episode, we gather resource management actions made by the neural network model and attach them to the knowledge replay buffer, which consists of at most 50000 latest resource management knowledge. At the end of an episode, each action taken is evaluated and their values are supplemented to corresponding items in the knowledge replay buffer. Therefore, our approach can be categorized as a Monte-Carlo method [62] instead of Temporal Difference method [3]. After finishing of each episode, if enough knowledge is accumulated in the knowledge replay buffer, neural network training process will be carried out with 1000 randomly selected samples from knowledge replay buffer.

Strategies in Accommodating and Improving RL based Approach

We briefly describe the special strategies and considerations we have taken in the process of better accommodating the reinforcement learning based model in the desired resource management problems as follows.

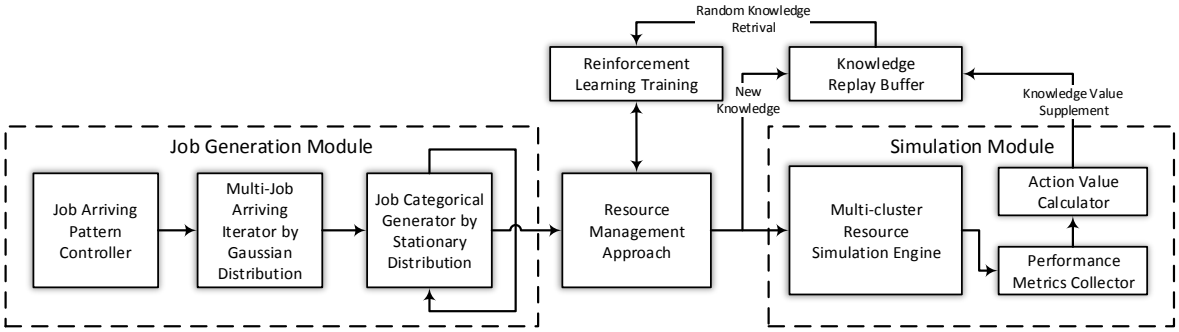


Figure 4.2: Reinforcement learning procedure in single episode.

Enabling resource management target selection among multiple applications

Note that for the designed neural network, it accepts feature vector of a single application awaiting resource allocation, and the output is the value estimation of all actions corresponding to allocating the aforementioned application to according cluster. However, it is possible that there are multiple applications awaiting scheduling by our resource manager at certain moments. We would like our reinforcement learning based approach to be able to deliberately choose the most suitable application among all awaiting ones at each time slice.

A fundamental requirement for neural network is the stationary length of input vector, which is obviously achievable when the input is with respect to one application, but not likely when input vector is used for describing multiple jobs, where the number of jobs remains variable and unbounded. Considering features of adopted reinforcement learning process, where an action is chosen based on the largest values given by all available actions, we decide to modify the outer-layer of the action selection process, instead of the neural network input vector, to accommodate the need in choosing among multiple awaiting applications.

In our modification, the neural network remains unchanged, that is, its input is still with respect to one single application. However, by utilizing the uniform meaning of values in the output, in each time slice, if there are multiple jobs awaiting, we would feed each of them to the neural network individually, then append all values together as one vector. The vector is then passed through the ε-greedy process as described below. For any position selection in vector, the position is then translated back to corresponding application and its action. This enables the choice among multiple awaiting jobs without modifying neural network input.

Improved epsilon-greedy strategy for more effective and efficient RL process

In reinforcement learning, there is a strategy with the name ε-greedy strategy. It can expand the exploration by randomly selecting actions with ε probability, so that the vision of the reinforcement learning is not restricted by current capability of the obtained model. It also provides opportunities to escape out of local optimum, and enhances chances to reach global optimum.

However, one disadvantage of the ε-greedy strategy is that it solely expands exploration by random actions, therefore no other possible valuable prior knowledge is utilized. This may hinder efficiency in achieving better evolved resource manager via reinforcement learning model. In light of this, we innovatively apply an improved ε-greedy strategy by default for more effective and efficient RL process. Specifically, the improved ε-greedy strategy uses both the random action and action from our baseline resource manager to increase exploration. The details of it are stated as below.

$$action = \begin{cases} \text{random action} & r_1 < \varepsilon_1 \ \ \& \ \ r_2 < \varepsilon_2 \\ \text{baseline guided action} & r_1 < \varepsilon_1 \ \ \& \ \ r_2 \geq \varepsilon_2 \\ \underset{j \in J, a \in A}{\arg\max} \ \mathrm{Q}_a(s_j) & r_1 \geq \varepsilon_1 \end{cases} \tag{4.3}$$

When deciding an action, a randomized value from 0 to 1 is compared to current ε_1 value. If less than, an action is selected with ε_2 possibility being random and $1-\varepsilon_2$ probability being the action selected by our baseline resource manager; otherwise, action with largest value given by network outputs is selected. Thus, knowledge from the baseline approach could be utilized to enhance reinforcement learning effectiveness and efficiency meanwhile the random exploration is carried out. This improved strategy is shown in Eqn. 4.3, where J is the set of scheduling awaiting jobs, A is the action set and $\mathrm{Q}_a(s_j)$ gives value estimation of action a for application j, r_1, r_2 are randoms in $[0,1]$. In experiments, we set $\varepsilon_2 = 0.5$, and ε_1 linearly decrements from 0.8 to 0.00001 in Episode 1 to Episode 1900 (total episode is 2000).

Training with randomized workloads

In specifying single application characteristics in a workload, there are multiple configuration parameters that can be varied, such as discrete resource utilization distribution for streaming application, single executor occupation time, execution deadline, and duration for application. These parameters define the execution features of an application. Whenever an application is generated, especially in verifying the performance effect of resource management approaches, we implement proper randomness to these parameters to expand generality of the workload.

The question remains whether in the reinforcement learning training process, application characteristic randomness, which enhances workload generality, will bring difficulty or even hinder the training process. We consequently verify the answer to this question in several circumstances and have made the following observation:

For different value function designs, it is possible that the broad generality of workload by application randomness may bring great difficulty in training process and harm the effectiveness of reinforcement learning model. Using one sampling of randomized workload as a fixed workload for

the entire training process could mostly alleviate the problem, and surprisingly, the resulting model ends up reasonably well for using in randomized workload environment. However, after refining the value function design to be what we present in this work, we discover that the effectiveness of the training process remains well even for randomized workload and the performance of the resulting model is further greatly improved comparing to the model obtained by training with a fixed sampling of randomized workload. The underlying reason behind this is apparent since the trained model with randomized workload has better knowledge and vision from the better workload generality. We thus apply training with randomized workloads in experiments.

Algorithm 3 Resource Management and Training Algorithm

```
i: Current episode; t: Time,
N: Total number of episodes,
NoW: Predefined number of applications in workload,
fNoW: Finished number of applications in workload,
RGA: Number of randomly Generated Applications,
vecV: Value vector for jobs in job pool,
nn: Neural network model,
kb: Knowledge replay buffer
kb=[]; initialize ε1
for i in range(N) do
  t=0; RGA=0; fNoW = 0
  while fNoW < NoW do
    if RGA < NoW then
      RGA += generateJobs(jobPool)
    end if
    vecV=[]
    for all job j in jobPool do
      generate feature vector s for j
      vecV.append(nn.eval(s))
    end for
    if random(0,1) < ε1 then
      if random(0,1) < ε2 then
        action=randint(0,len(vecV)-1)
      else
        action=action_baseline
      end if
    else
      action=argmax(vecV)
    end if
    takeAction(action)
    knowl=generateKnowledge(action)
    kb.push(knowl)
    fNoW += removeFinishedJobs()
    t++
  end while
  update value for new knowledges in episode i
  if len(kb) > batchsize then
    minibatch=random.sample(kb,batchsize)
    Train neural network by stochastic gradient descent
  end if
  ε1 decrements
end for
```

RL Training Algorithm for Resource Management

The main algorithm for RL training is presented in Algorithm 3. A detailed presentation of the procedure for one single episode is shown in Figure 4.2. The entire training process is composed by N training episodes. In our experiments, $N = 2000$. Obtained knowledge in each episode is pushed into the knowledge replay buffer. At the end of each episode, the value of each new knowledge is calculated and supplemented to the knowledge buffer. If sufficient knowledge exists, the neural network training is launched where the newly obtained model is used in the next episode. The weight of the neural network is updated via Stochastic Gradient Descent (SGD) with mean square error.

Experiment Results

In this section, we present experiment design and results. The experiment is launched in our designed simulator, in a computing resource of 5 clusters with different number of executors defined by $[500, 800, 1200, 1300, 1900]$. For inner-cluster local scheduler, a First-in-First-out (FIFO) scheduler is adopted, which is popularly provided as a default scheduler in various big data platforms. First, we introduce design of job arriving patterns and the baseline rule-based resource managers. Then we present experimental results regarding the performance comparison of RL based resource management approach to baseline models.

Job Arriving Patterns

To thoroughly testify the effectiveness of designed reinforcement learning based approach, we adopt three statistical patterns as job arriving patterns in resource management experiments.

By Bernoulli process

In the first job arriving pattern, we assume at each simulation time step, a job arriving event happens with a stationary probability $\rho = 0.08$, where each experiment is fully independent. This obeys the definition of "*Bernoulli process*". When such an event happens, we let the number of arriving jobs be decided following a rectified discrete Gaussian distribution: $Num_{job} = \max(\text{round}(\text{N}(\mu, \sigma^2)), 1)$. For Bernoulli pattern, $\mu = 1.5,\ \sigma = 1$, for other two patterns, $\mu = 3,\ \sigma = 1$.

The category of arriving jobs also follows a stationary distribution, with β_1, β_2, and β_3 being probabilities for regular non-time-critical, non-streaming time-critical and streaming time-critical, respectively. Here $\beta_1 + \beta_2 + \beta_3 = 1$, and remains the same for other job arriving patterns as well. For all three patterns, $\beta_1 = 0.5,\ \beta_2 = 0.25$, and $\beta_3 = 0.25$. From the property of Bernoulli process, the probability of having a new job arriving event with interval i is equivalent as the probability of having a first success in i consecutive yet independent Bernoulli experiments, of which the expression can be written as: $P(i) = (1-\rho)^{i-1} \cdot \rho \cdot I_{i>0}(i)$. Here $I_{set}(x)$ is the indicator function, it is 1 when $x \in set$, otherwise 0.

By Uniform distribution

In the second job arriving pattern, we assume the occurring interval of job arriving event follows a discrete uniform distribution in $[a,b]$, here $a = 1,\ b = 39$. That is, in our selected range, the occurring probability for job arriving event with each interval time i is the same. Thus, the probability function can be presented as:

$$P(i) = \frac{1}{|b-a+1|} I_{i \in [a,b]}(i) \tag{4.4}$$

By Beta distribution

In the third job arriving pattern, we assume the occurring interval of job arriving event follows a modified discrete version of Beta distribution. α, β and M are pattern parameters, where in our experiment $\alpha = 4,\ \beta = 2,\ M = 30$. The probability of job arriving event with interval i therefore can be written as:

$$P(i) = \frac{I_{i>0}(i) \cdot \Gamma(\alpha+\beta) \cdot [B\left(\frac{i}{M};\alpha,\beta\right) - (B\left(\frac{i-1}{M};\alpha,\beta\right)]}{\Gamma(\alpha)\Gamma(\beta)} \tag{4.5}$$

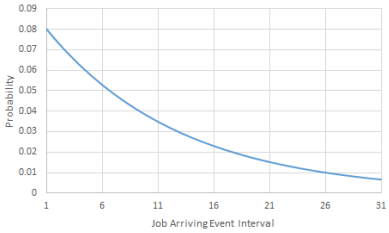
where,

$$B(x;\alpha,\beta) = \int_0^x t^{\alpha-1}(1-t)^{\beta-1}dt \tag{4.6}$$

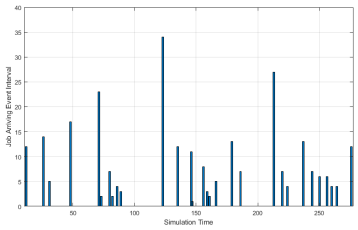
and $\Gamma(x)$ is defined as the Gamma function:

$$\Gamma(x) = \int_0^\infty t^{x-1}e^{-t}dt \tag{4.7}$$

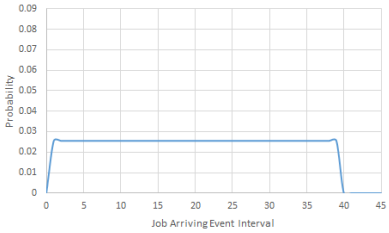
The probability of job arriving event with interval time i for all three job arriving patterns, and corresponding sampling of first 30 job arriving events in an episode, could be seen in Figure 4.3.



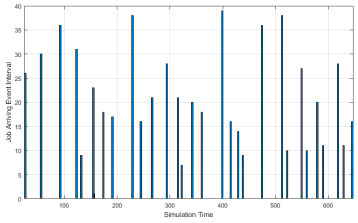
(a) Bernoulli pattern probability



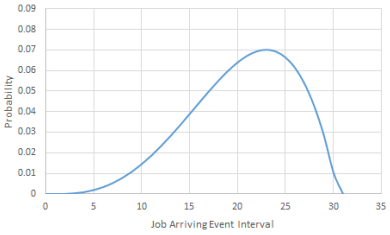
(b) Bernoulli pattern sampling



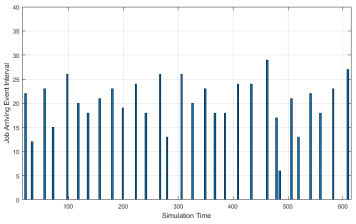
(c) Uniform pattern probability



(d) Uniform pattern sampling



(e) Beta pattern probability



(f) Beta pattern sampling

Figure 4.3: Job arriving pattern probability and pattern sampling

Rule-based Baseline Resource Managers

To compare with the effectiveness and justify the necessity of the proposed reinforcement learning based approaches, we put forward rule-based resource management approaches as the baseline models. These different rule-based resource management methods are designed in purpose of expanding baseline solution generality. During designing, we at best effort design competitive

ruled-based solutions that utilize available information reasonably and at its utmost; meanwhile keeping variance and generality of solutions in mind.

The rule-based approaches are shown as follow:

- *Random action (Random)*: Both the target job when multiple jobs are awaiting scheduling and the scheduling action for this job are selected randomly.
- *Smallest first-P (SF-P)*: When multiple jobs are awaiting scheduling, the job with smallest computing capacity requirement will be selected. This scheduler will examine cluster utilization percentage in the nearest past 100 time slices, and select the one with the lowest average percentage as the destination for the target job.
- *Largest first-P (LF-P)*: Same as previous, except that the job with largest requirement in computing capacity is selected among multiple awaiting jobs.
- *Smallest first-E (SF-E)*: *SF-E* is the same with *SF-P* in selecting target job among multiple awaiting ones. But it will examine cluster resource utilization in the nearest past 100 time slices, and select the one with the largest average number of available executors as the destination for the target job.
- *Largest first-E (LF-E)*: Same as previous, except that the job with largest requirement in computing capacity is selected among multiple awaiting jobs.

Evaluation Metrics

Whenever evaluating approaches, all participating approaches will be tested for 50 independent testing episodes. Each testing episode is independent in the sense that its workload consisting of 500 jobs is entirely randomly generated following designated job arriving pattern. However, in

each testing episode, this same workload of 500 jobs are submitted to all approaches. To enable performance comparison, we present multiple evaluation metrics as follow.

Firstly, as previously stated, there are two major performance factors we take into consideration as shown below.

- *TMDL*: Total occurrence of missing temporal deadline events in all clusters of the overall computing resource during one episode, with respect to the resource management approach.
- *AJDR*: Average job delay ratio is defined as the average job running overhead percentage for all 500 jobs in one episode with respect to certain resource management approach. Specifically, it is defined as:

$$AJDR = \sum_{i=1}^{J} \left(\frac{100 \cdot \sum_{j=1}^{N_i} \left(\frac{AR_{ij}}{OR_{ij}} - 1 \right)}{N_i} \right) / J \tag{4.8}$$

  where $J = 500$ is the total number of jobs in one episode. N_i is the total repeating runtime of job i, it is 1 for non-streaming application and larger than 1 for streaming applications. AR_{ij} and OR_{ij} are the actual running time and optimal expected running time of job i in its j-th running, respectively.

Then, besides these two direct metrics, to evaluate system performance in a more integrated and well-rounded way with considering both performance metrics as mentioned before, we include one more quantitative measurement and four more comparative methods.

Quantitative Measurement

The quantitative measurement is presented in Eqn. 4.9. It is designed in purpose of concisely evaluating both major performance metrics TMDL and AJDR in combination. With the reciprocal

operation, the eventual $Eval_{app}$ can be interpreted as an evaluation score where higher score implies better performance, with 0 being the lower bound. The weights in linear combination are chosen considering data scales in separate metrics.

$$Eval_{app} = (0.02 \cdot \text{TMDL} + \text{AJDR})^{-1} \tag{4.9}$$

Comparative Methods

Comparative methods will be used in different configurations for comparing proposed RL approaches with baseline approaches (the best of candidates is chosen) in 50 testing episodes of workloads. The methods are constructed based on deciding *win*(1), *lose*(0), and optionally, *even*(0.5) for proposed RL approach in each testing episode. The sum of which is then multiplied by 2 to convert into a 100 basis. At all times, $Score_{RL} + Score_{Base} = 100$, with a winning-for-all-rounds RL approach during testing scored at 100. Difference in score definition decides its strictness.

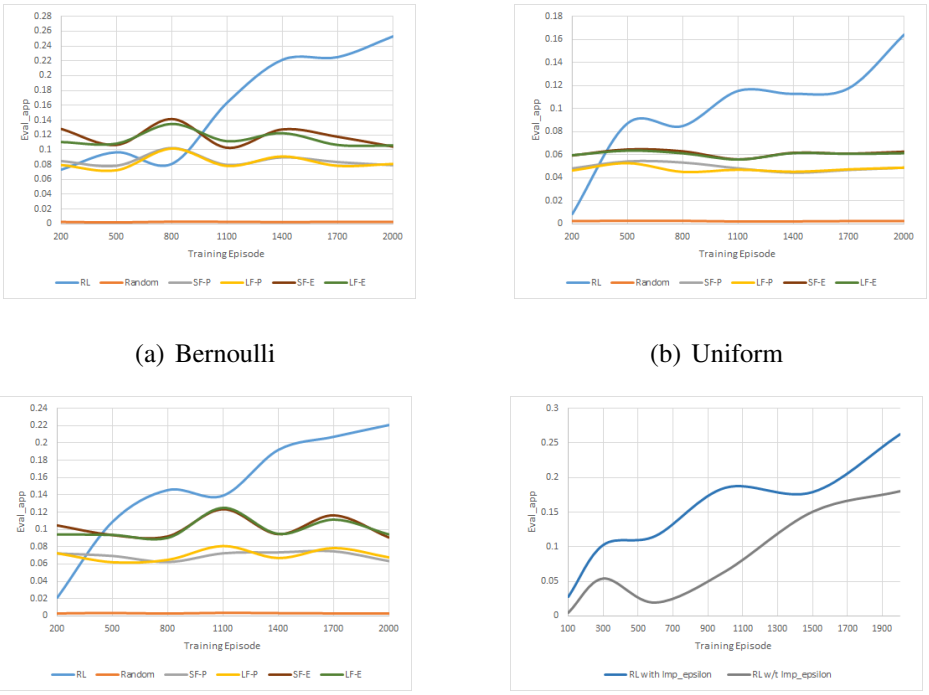
- Score-A: If RL approach outperforms the baseline in **both** metrics in a testing episode, it scores 1, otherwise 0. It is a very strict evaluation standard for RL approach, in purpose of showing absolute dominant percentage of RL approach over baseline.
- Score-B: Same as Score-A, but additionally adds the 'even' case, that is, the RL approach receives 0.5 if it outperforms in only one of either metrics.
- Score-C: RL receives 1 if its evaluation $Eval_{app}$ is higher than that of the baseline solution, calculated by Eqn. 4.9. Otherwise, it receives 0.
- Score-D: The percentage changes respectively in TMDL and AJDR from baseline to RL approach are computed and added together. If the overall percentage change is negative (thus implies improved performance for RL approach), RL gets 1, otherwise 0.

Performance Comparison

In this section, we present performance comparison of RL approach with multiple baselines for all job arriving patterns. The RL approach is constructed by description in Section 4, with employing all modification strategies mentioned in Section 4 (SF-E as baseline in improved ε-greedy method). Each RL approach is trained for 2000 episodes.

For each job arriving pattern, we present performance comparison of RL with other baseline approaches along different training episodes. Following by quantitative metric and score comparison of RL at final training episode with the best baseline approach.

Furthermore, we present the comparison of RL approaches with and without utilization of our improved ε-greedy strategy in reinforcement learning process. We also present the result of applying the obtained RL approach to workloads with intentionally varied computing capability requirement statistics, which demonstrates the generality of our obtained RL resource management approach.



(a) Bernoulli

(b) Uniform

(c) Beta

(d) RL comparison

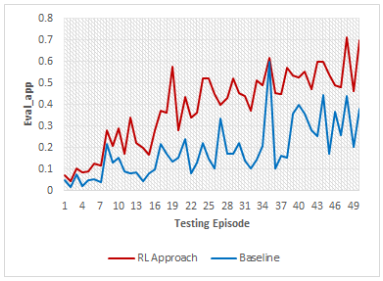
Figure 4.4: (a-c) Performance comparison of RL approach and different baselines for Bernoulli, Uniform and Beta job arriving pattern respectively in different training episodes. (d) Performance comparison of RL approach with and without our improved ε-greedy method in different training episodes.

For three job arriving patterns, the performance comparisons of RL approach with different baselines, with respect to training episodes are shown in Figure 4.4(a), 4.4(b) and 4.4(c). When computing $Eval_{app}$ in Figure 4.4, TMDL and AJDR are averaged respectively over 50 testing episodes. For all job arriving patterns, it is observable that among five baselines, the Random baseline performs the worst; SF-P and LF-P although perform slightly differently, are on average at a similar level; as well, SF-E and LF-E perform at a very similar level with slight differences occasionally.

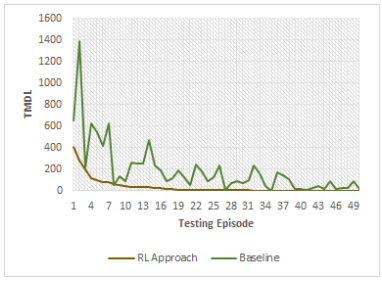
From Figure 4.4(a), 4.4(b) and 4.4(c) we can see that our proposed RL approach for all job arriving patterns gradually improves itself, surpassing all opponents in early training episodes, and eventually achieves big performance advantage over all baseline approaches. This fulfills our desire in achieving good resource management approach.

When further looking into the final model, which is the RL approach at final training episode, we first select its best opponent. By examining the overall performance of all baseline approaches, in all three job arriving patterns, SF-E remains performing as the best baseline, it is thus selected for pairwise comparison with final RL approach for all three job arriving patterns.

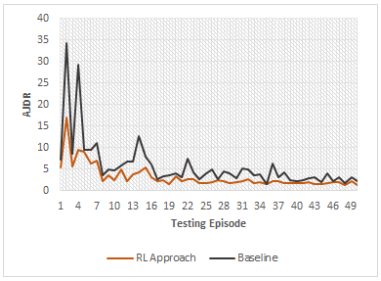
The comparison of corresponding RL approach (at final training episode) and best baseline SF-E in 50 testing episodes for three job arriving patterns can be found in Figures 4.5. Specifically, for Bernoulli pattern: 4.5(a), 4.5(b), 4.5(c); for Uniform pattern: 4.5(d), 4.5(e), 4.5(f); and for Beta pattern: 4.5(g), 4.5(h), 4.5(i), which are all related to $Eval_{app}$, TMDL and AJDR metrics, respectively. It is worth mentioning that for $Eval_{app}$ metric, the **higher** value means better performance. Whereas for the later two metrics TMDL and AJDR, the **lower** means the better performance. It is apparent that our final RL approach consistently performs very well in all three job arriving patterns, and outperforms SF-E in all three metrics with significant difference.



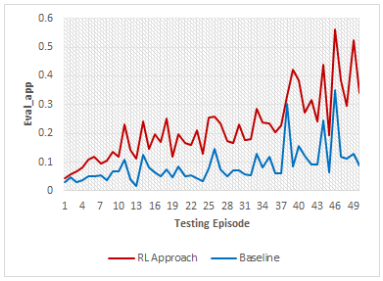
(a) Bernoulli $Eval_{app}$



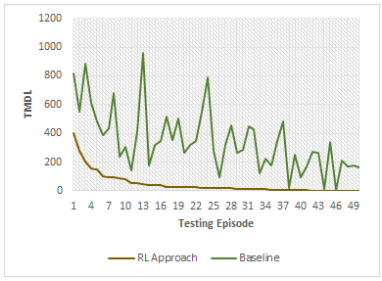
(b) Bernoulli TMDL



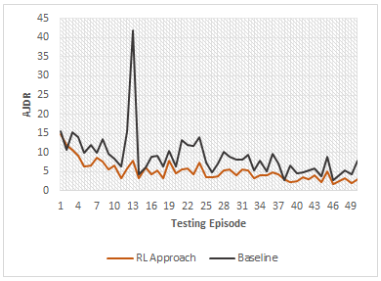
(c) Bernoulli AJDR



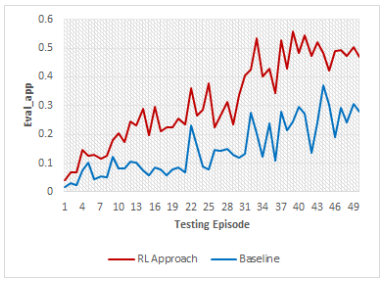
(d) Uniform $Eval_{app}$



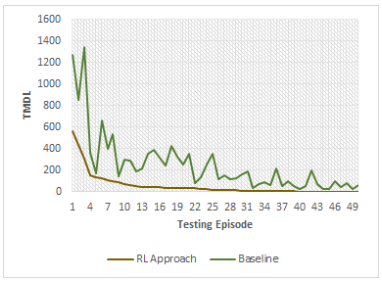
(e) Uniform TMDL



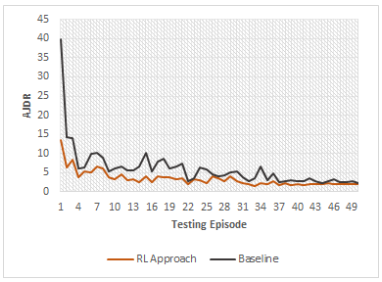
(f) Uniform AJDR



(g) Beta $Eval_{app}$



(h) Beta TMDL



(i) Beta AJDR

Figure 4.5: Comparison of RL (at final training episode) with the best baseline (SF-E) for $Eval_{app}$, TMDL and AJDR metrics in different job arriving patterns. Graphs are showing for 50 testing episodes used for comparison, sorted by RL TMDL in convenience of viewing. Three rows correspond to *Bernoulli*, *Uniform* and *Beta*, respectively. Three columns correspond to $Eval_{app}$, TMDL and AJDR, respectively. For $Eval_{app}$, the **higher** the better. For TMDL and AJDR, the **lower** the better.

The average statistics of aforementioned 50 testing episodes with respect to all three job arriving patterns can be found in Table 4.2:

Bernoulli pattern: For TMDL, the average belongs to RL and Baseline are 35.04 and 189.34. RL approach achieves to reduce TMDL by a significant ratio of 5.40. It also reduces AJDR by ratio 1.78. For four scoring metrics, it suffices to say that even for Score-A, the most strict standard for RL, it achieves 98 out of 100. That is, according to testing, it can dominant baseline in both TMDL and AJDR simultaneously with approximately 98 percentages probability. Other scores are even higher.

Uniform pattern: For TMDL, RL approach achieves to reduce TMDL by a significant ratio of 7.55, it also receives reduction in AJDR by 2.08. For four scoring metrics, the scores keep showing RL approach as dominant solution, with Score-A, the most strict one, being 96 out of 100.

Beta pattern: Once again, RL approach achieves to reduce TMDL by a good ratio of 4.40, and receives reduction in AJDR by 1.79. For four scoring metrics relating to 50 testing episodes, all scores are identically 100.

By examining performance comparison, we are therefore confident to consider achieved RL approach as a good resource management approach for designated scenario and it becomes a much better resource management solution than aforementioned rule-based baselines.

After showing major performance comparison, we would like to supplement additional experiments. Firstly, we want to verify influence of the improved ε-greedy strategy described in Section 4. We use Bernoulli process pattern and compare the performance of obtained RL approach with and without the improved ε-greedy method as shown in Figure 4.4(d). The improved ε-greedy method (with SF-E as baseline) indeed improves both the effectiveness and efficiency of RL approach. The RL approach with improved ε-greedy strategy gains much better final performance. And even at

middle stage of RL process, it already achieves comparable performance to the one at final episode without the improved ε-greedy method. This justifies the necessity in utilizing proposed improved ε-greedy strategy.

Secondly, although our RL resource management approach demonstrates good generality by being suitable to randomly generated hybrid workloads in multiple job arriving patterns, we intend to further apply stresses to the obtained RL approach by the following experiments:

Table 4.2: Performance comparison of RL approach and SF-E for three different arriving patterns

Metric	**Bernoulli arriving pattern**			**Uniform arriving pattern**			**Beta arriving pattern**		
	RL Approach	**Baseline**	**Ratio**	**RL Approach**	**Baseline**	**Ratio**	**RL Approach**	**Baseline**	**Ratio**
TMDL	35.04	189.34	5.40	46.34	349.9	7.55	56.52	248.86	4.40
AJDR	3.24	5.78	1.78	5.18	10.80	2.08	3.40	6.07	1.79
Score-A	98	2	49	96	4	24	100	0	∞
Score-B	99	1	99	98	2	49	100	0	∞
Score-C	100	0	∞	100	0	∞	100	0	∞
Score-D	100	0	∞	100	0	∞	100	0	∞

For an already obtained RL model, we vary several statistical characteristics during workload generation in testing. The newly randomly generated workloads, although following the same job arriving pattern as in training, show significantly different computing capability desires than ones during training. This consequently brings challenges to RL approach generality. Using uniform distribution job arriving pattern, we generate two new testing workload patterns, "eased" uniform and "stressed" uniform, where randomly generated jobs have statistically **less** (**more**) computing capability requirement than original ones respectively. We test obtained final RL approach by original uniform pattern and SF-E against new workload patterns. The result is shown in Table 4.3.

As expected, comparing to original version, it can be seen that the performance of RL and SF-E in

the sense of TMDL and AJDR both simultaneously improve (deteriorate) in the "eased" ("stressed") version, respectively, due to changes in computing capability requirement statistics. However, regardless of the pattern change, the originally obtained RL approach remains performing very well in either cases. It means that the generality of our RL approach is further fortified. And this concludes our experiment section.

Table 4.3: Performance comparison of RL and SF-E for Uniform arriving pattern with eased and stressed workloads

Metric	**Eased workloads**			**Stressed workloads**		
	RL	**Baseline**	**Ratio**	**RL**	**Baseline**	**Ratio**
TMDL	16.36	140.52	8.59	350.44	1027.86	2.93
AJDR	3.12	4.87	1.56	13.76	19.96	1.45
Score-A	96	4	24	92	8	11.5
Score-B	98	2	49	95	5	19
Score-C	100	0	∞	94	6	15.67
Score-D	100	0	∞	96	4	24

Summary

In this chapter, we analyze reinforcement learning based approach for resource management of hybrid workloads in large-scale distributed computing environment. By comparing performance with baseline solutions in various job arriving patterns, as well as comparing to other RL approach version, we demonstrate the effectiveness and generality of obtained RL approach. It is observed during testing that the TMDL metric is improved by up to 7.55 times, while the AJDR metric is improved by up to 2.08 times. In conclusion, we successfully obtain better RL based resource management approaches for hybrid workloads in distributed big data computing environment.

CHAPTER 5: ELASTICITY-COMPATIBLE SCHEDULING FOR TIME-CRITICAL COMPUTING IN HETEROGENEOUS ENVIRONMENTS

[1] Nowadays' distributed computing resources hosting data analytical jobs are often organized in the unit of cluster, where each cluster represents a closed association of computing nodes that a data analytical work can be carried out in a distributed manner [82]. To further expand the computing resources in a large scale, opting for a multi-cluster computing resource raises its benefit and necessity due to the following reasons: Firstly, the multi-cluster environment can be naturally derived from the geographical distribution of resource. Secondly, it sometimes can be beneficial to organize overall resource into clusters as a separation for managing jobs and data. Thirdly, the facility may utilize online computing resources from service providers as additional clusters for resource expansion. Examples of a multi-cluster environment could happen when an institution has several physical clusters at its branches (may at different locations). Another example could be a hybrid-cloud, which may consist of multiple private (self-owned) and public cloud clusters. In accordance, an efficient resource management being aware and compatible with such a multi-cluster computing infrastructure should be presented to guide job distribution.

Moreover, there are other computing environment features to be taken into considerations. Firstly, it is possible that computing nodes in different clusters have different computing capabilities. In other words, multiple clusters composing the environment could be heterogeneous. Secondly, certain clusters could possess elasticity. Here, elasticity is referring to that the capacity of the cluster could be temporally expanded as desired to fit the expanded computing demands, which is more often observed in cloud computing as one of its main features [67]. It is certainly beneficial if a proposed

[1] Content of this chapter is based on our published paper [50].

resource manager could be compatible with such elasticity capability and be aware of heterogeneity.

Furthermore, job features are equivalently important. For example, besides general analytical jobs, many jobs nowadays could be streaming jobs, which aim to timely process gradually arriving streaming data, and can be regarded as conducting repetitive executions with different batch of data. Such jobs are intrinsically end-to-end delay-sensitive, thus are time-critical. In a multi-cluster environment, considering differentiated transmission overhead for a time-critical job, it requires differentiated temporal execution deadline on different clusters, which should be regarded as a job feature and be considered by the resource management approach.

All computing environment features such as multi-cluster, elasticity and heterogeneity, and the job features such as job type and timeliness, add up to the complexity of generating a satisfactory resource management approach that can well utilize the multi-cluster environment. Thus, it is very hard to devise a comprehensive rule-based approach. Moreover, the high dynamic feature of the system status as well as the desire for a timely online scheduling decision hinder the utilization of iterative searching based approaches. In vision of this, we propose to utilize deep reinforcement learning (DRL) techniques in this work to obtain a resource management that could fulfill the expectation. The major contributions include:

- We propose a deep reinforcement learning based approach utilizing **LSTM model** and **multi-target regression with partial model sharing mechanism**, and compare its effectiveness with respect to other baseline and RL approaches.
- The DRL-based resource management is designed for distributed multi-cluster computing environments with considering its **heterogeneity** and being **elasticity-compatible**.
- The DRL-based resource management provides scheduling support for **time-critical** (delay-sensitive) computing in such a multi-cluster environment as described.

Problem Description

The intended global resource management is aiming to **(1) reduce occurrences of missing temporal deadline events** while **(2) maintaining a low average execution time ratio** for a hybrid workload containing multiple time-critical and general jobs, by properly scheduling them to appropriate computing clusters in the underlying computing environment. We depict the overall architecture of the problem in Figure 5.1 and present detailed problem description as follows.

The underlying computing environment is composed of multiple computing clusters. For each cluster, its overall computing resource is expressed as the number of executors it could provide. The “executor" here is a basic resource allocation unit containing a combination of virtual CPU and memory resources, which is adopted in popular resource managing frameworks such as YARN [82], and utilized by computing environments such as Apache Hadoop and Spark. Different from [51], where executors are assuming to be identical among different clusters, cluster heterogeneity is allowed in this work. That is, different clusters in the computing environment may have diversified executors with different computing capabilities. A heterogeneity factor (the larger the stronger) will be assigned to each kind of such executors, representing its relative computing capability. Furthermore, in this work, clusters are allowed to have elasticity, that is, their computing capability in terms of executor numbers, can be temporarily expanded (with an upper bound) when necessary, to fit workload pressure. This assumption coordinates with the trending of elastic computing resources.

The holistic workload contains a number of continually arriving jobs, where each job can be classified into one of the three categories[51]: **streaming jobs** (Cate-1), **non-streaming time-critical jobs** (Cate-2) and **other general non-time-critical jobs** (Cate-3). As aforementioned, streaming jobs can be regarded as conducting repetitive executions with data batches and are inherently delay-sensitive therefore time-critical. Such a categorization of jobs helps manage hybrid

workload in presenting a large range of representative user analytical needs.

Arriving jobs first enter the global job buffer. For each moment t when global job buffer is non-empty, given one job in the global buffer, the intended global resource manager selects a proper cluster in the computing environment for execution. Here, how to derive an effective approach to fulfill the global resource management is the main problem.

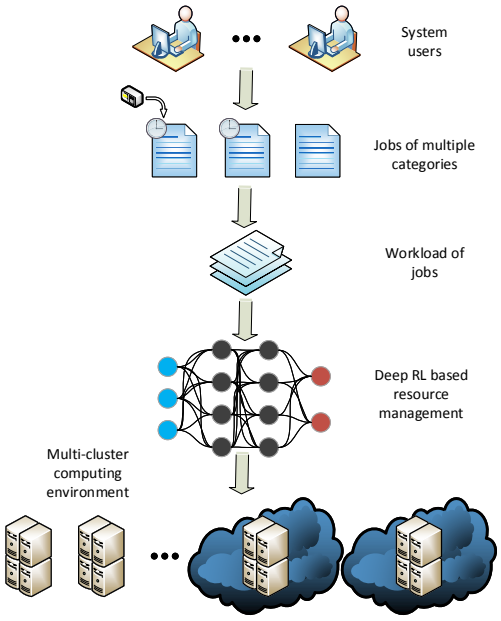


Figure 5.1: An illustration of the problem architecture.

This problem is inevitably complex. Firstly, abundant information, such as job and computing environment features and system dynamic need to be considered. Secondly, a job can have cluster-specific difference, such as cluster-specific deadlines due to differentiated transmission overhead and cluster-specific execution time due to cluster heterogeneity. Cluster elasticity could also change system resource status and thus affect job execution. The model should be able to handle these issues and provide good performance to both management goals. Solely attending to partial of the available information or partial of the objectives can lead to unsatisfactory performances. Integrated

balancing between goals and short-term sacrificing yet long-term benefiting scheduling actions are potentially desired. The extreme difficulty lays in whether, when or how such kind of trade-off should be made, which is quite uncertain for rule-based models and brings favor to DRL-based approaches.

The Deep Reinforcement Learning Based Approach

Our goal is to obtain an efficient resource management approach with a neural network model via deep reinforcement learning. In this section, we state our elaborative considerations in accommodating the problem to model design and training skills, starting by a brief introduction to reinforcement learning technique.

Introduction to Reinforcement Learning

Reinforcement learning is a mechanism that enables model improvement through continual interaction with the application environment defined upon several key concepts: environment, state, episode, action and reward (or value).

Environment describes the overall world where the actual state transitions happen. **State** is used to express the environment status at a moment. Based on the adopted representation, the state s_t at moment t can potentially contain historical or instantaneous information of the environment. **Action** represents the set of actions performable to environment at the moment. The instant **reward** is a quantitative incentive feedback that the model receives from environment at a moment, whereas the **value** is a form of the accumulated reward (optionally decayed) observed in a longer duration. Definition of the value of taking an action a in state s under a decision-making policy π can be

formulated as below [78]:

$$V^{\pi}(s,a) = E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^{k} r_{t+k+1} \mid s_t = s, a_t = a\right\}$$

where E_{π} represents the expectation under policy π, r_{t+k+1} is the instant reward at moment $t+k+1$, a_t is the action taken at moment t and γ is the decay factor. An **episode** is a round of 'game' that marks the reaching of the termination state.

Reinforcement learning enables model improvement via the general interaction mechanism as depicted in Figure 5.2: Based on environment state s_t at t, an action is decided by the model and an environment state transition consequently happens. Over the time, model improvement can be carried out by learning from collected interactive experience in purpose of maximizing action values. Such a process is repeated multiple times until the accomplishment of training process.

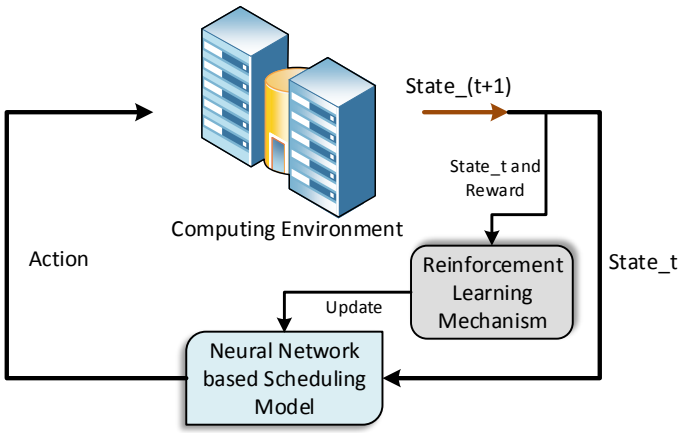


Figure 5.2: An illustration of reinforcement learning.

Reinforcement Learning Method Design

In this work, **environment** is the entire computational system where the overall processing of jobs take place. The **action** set contains number of actions equal to the number of clusters forming the computing infrastructure, where each action corresponds to deploying the job to that specific cluster. An **episode** is the whole process of scheduling and finishing the execution of a workload consisting of a certain number of jobs.

Environment **state** is composed of two main components: computing system state and scheduling job information. Computing system state (iterative for each cluster) includes static cluster features like its sequence number, elasticity information (including cluster's normal capacity and maximum capacity under expansion), heterogeneity factor (hetero factor) and dynamic features like cluster occupation status in a past time interval and current accumulated total occurrences of missing deadline events in cluster.

For scheduling job information, jobs in different categories can possess different attribute sets. We utilize streaming job (Cate-1) information as a super set for description, and other job categories can adjust to this with unambiguous accommodations. Such information includes job category, expected hetero factor, standard execution time with respect to using executors with the expected hetero factor, execution deadline, total duration, discrete resource request distribution and its heterogeneity sensitivity measure to describe job's adaptation capability to clusters with different hetero factors. We explain some of the keywords here as below:

Heterogeneity sensitivity measure: A job's execution time is often proportionally affected by cluster's hetero factor if the factor is within a given range. In this work, a job's heterogeneity sensitivity measure '*sen*' (integers in $[1,5]$ in experiment) is used to represent such range by $R = [h - sen \cdot \zeta, h + sen \cdot \zeta]$, where h is the job's expected hetero factor and $\zeta = 10$ in experiment.

If the cluster's hetero factor belongs to R, this job's execution time in cluster will be proportionally affected from its standard execution time by the cluster hetero factor. Otherwise, if the cluster's hetero factor is outside the range, this job will cease to further fully reflect change in its execution time. This corresponds to job behaviors in practice where its execution time could be affected accordingly by executors within a range of hetero factors, yet will not fully receive benefit (or harm) in execution time for too large hetero factor changes, due to other execution overheads.

Job's cluster-specific difference: Job can have cluster-specific difference. Such as, job's execution deadline for each cluster can be different, possibly due to cluster-specific data transmission overheads from data source, when regulating a uniform end-to-end batch execution time. It could also be due to job's program transmission overhead to cluster before starting the execution. Job execution time can also be cluster-specific due to cluster heterogeneity. To lower user profiling burden, instead of requiring cluster-specific execution times, our model only requires the standard execution time with respect to (w.r.t.) job's expected hetero factor and can handle heterogeneity even without the complete execution time information.

Discrete resource request distribution: A streaming job may have resource request fluctuation during its repetitive executions. Analogous to [51], when executing in a cluster, it is affiliated with an length 10 array to describe its discrete resource request distribution, with each position accounting for 10 percentage possibility. For example, if such a distribution array containing eight 30s and two 60s, it means in each execution of the job, it has 0.8 probability to request 30 executors and 0.2 probability to request 60 executors in this cluster.

The overall composition of the state representation is presented in Table 5.1. For cluster occupation status, we would like to use the records roughly in the latest 100 steps. In experiment, the latest 105 steps is used per input design desire, and this information is specially traversed to create a 150 dimensional vector for each cluster. Details of the traverse will be stated in Section 5.

Table 5.1: State representation in our deep reinforcement learning model for 5 clusters

Vector Component	**Dimensions**
Cluster ($i = 1 \ldots 5$)	
Cluster sequence number	1
Normal capacity	1
Maximum capacity if elastic	1
Cluster heterogeneity factor	1
Occupation status (latest 105 steps)*	$105 \rightarrow 150$
Current total missing deadlines	1
	775 $(\mathbf{155 \times 5})$
Job	
Category	1
Expected heterogeneity factor	1
Heterogeneity sensitivity	1
Discrete resource request distribution	10×5
Standard execution time	1×5
Execution deadline	1×5
Duration	1
	64
Overall state vector	**839** $(\mathbf{775 + 64})$
Only * row includes temporal information	

The influence of deploying a job onto a cluster is destined to be long-term due to its execution duration. Meanwhile, since performance metrics related to number of missing deadline events and execution time statistics have significant response delay from occurring moments of their most influential contributing factors such as inappropriate deployment or resource competition within cluster, it is difficult to define the instantaneous action reward at any moment. In this vision, we alternatively define the value of an action, concentrating on action's long-term influence. In this work, the **value** $v^{(j)}$ of an action that schedules a job j to a cluster at some moment, is defined as follows:

$$v^{(j)} = \frac{\eta_c \cdot m_{ih} \cdot m_{ic}}{-\eta_j} \left[M_j + \sum_{t=t_s}^{t_e} \beta^{D_t} \left(W_j^{(t)} + W_{cl}^{(t)} \right) \right] - \psi_{ih} \cdot \psi_{ic} \cdot R_j$$

Here, η_c and η_j are the heterogeneity factor of the cluster and the expected heterogeneity factor of

the job, respectively. M_j is the number of missing deadline events of job j where resource waiting does not happen at execution beginning. Note that a job can have more than one missing deadline events, such as, since a streaming job can be repetitively executed for multiple times. $W_j^{(t)}$ records the happening of each missing deadline event of job j at moment t, if it does not belong to M_j. Similarly, $W_{cl}^{(t)}$ records the number of missing deadline events of all jobs in the cluster at moment t if resource waiting happens at their execution beginning. t_s and t_e correspond to the deployment and termination moment of job j, respectively. β is the decay factor, and D_t records how many new jobs have been deployed to the cluster after t_s, till moment t. R_j represents the overall average execution delay ratio of job j (its average execution time divided by standard execution time, then minus 1 for representing delay). In the experiment, weighted factors are applied to the components of $v^{(j)}$, we omit its showing in the formula for simplicity.

m_{ih} and m_{ic} follow Eq. 5.1 of m_Ω, where Ω services as a place holder for 'ih' or 'ic'. The events corresponding to 'ih' and 'ic' are Improper_Heterogeneity (IH) and Initial_Competition (IC), respectively. Here, IH refers to the case where an improper cluster arrangement is caused by the scheduling action, that the cluster heterogeneity factor is too low for the job, causing execution deadline violation. IC refers to where resource competition (resource waiting) happens right after the job's deployment to the cluster designated by the scheduling action. Similarly, ψ_{ih} and ψ_{ic} follow Eq. 5.1 of ψ_Ω, respectively. Here, $P_{m_\Omega} \geq 1$ and $P_{\psi_\Omega} \geq 1$.

$$m_\Omega = \begin{cases} P_{m_\Omega} & \text{isEvent} \\ 1 & \text{Otherwise} \end{cases} \quad \psi_\Omega = \begin{cases} P_{\psi_\Omega} & \text{isEvent \& } R_j > 0 \\ 1/P_{\psi_\Omega} & \text{isEvent \& } R_j \leq 0 \\ 1 & \text{Otherwise} \end{cases} \tag{5.1}$$

The design of the value formula is originated from the consideration that when resource competition contributes to the causing of the missing deadline event of job j, the root cause of the resource

competition can be complex. It could be due to the deployment of job j on a specific cluster or later deployment of other jobs on the same cluster. To alleviate such intertwined influence when deriving a clearer action evaluation, all missing deadline events of job j related to resource competition will be decayed over numbers of newly arriving jobs after j. Furthermore, the number of overall missing deadline events at any moment from all jobs in the cluster which could be caused by resource competition is also added following the same decay pattern, in purpose of attending the potential mutual influence of j from and to other coexisting jobs in the cluster. On the contrary, other missing deadline events of job j without resource waiting will not be decayed over time. The factor $\frac{\eta_c}{\eta_j}$ in formula helps the model discern the potential differentiated effects of cluster heterogeneity to the job. Such heterogeneity influence is also considered in R_j due to its definition. The factors m_{ih}, m_{ic}, ψ_{ih} and ψ_{ic} work as a whole to assist the proper avoidance of certain irregular behaviors which the model should avoid.

DRL Model Structure and Decomposition of Value Definition

According to Table 5.1, the model input (RL system state) contains two types of information: **Non-temporal information** describing static or accumulated properties of job and computing environment, and **temporal information** related to computing environment occupation status in an interval of past moments. There exist neural network structures more specialized in dealing with temporal sequential information, such as Long Short-Term Memory (LSTM). LSTM has shown effectiveness in various tasks, such as speech recognition, music modelling and language translation [25] [90]. We plan to utilize the LSTM structure to process the temporal information portion of the input.

Also, the action value is a vital feedback signal for RL process. In fact, the neural network in our approach is directly performed as a value estimator. How well the action value can be

estimated will undoubtedly affect performance in a large extent. Nonetheless, the value definition contains multiple components for better depicting action influence and such constitution adds up the estimation complexity. It makes the changing behavior of the integrated action value more difficult to be predicted. And, the possibly differentiated numerical range of components and their variation patterns could vanish influence of some individual component, which may affect learning and the multi-goal optimization result.

A plausible solution is to consider components in the value definition as multi-objectives in reinforcement learning, converting it to a Multi-Objective Reinforcement Learning (MORL) problem [81]. In a MORL problem, the value space can be composed of multiple dimensions, *i.e.,* $\mathbf{V}^{\pi}(s,a) = \left(v_1^{\pi}(s,a), \ldots, v_m^{\pi}(s,a)\right)$. With such formulations, our approach can be described as a scalarized[80] single-policy[23, 58] learning algorithm for MORL. More specifically, a linear scalarization function is utilized to regulate a united measure over the vector of the value components, and provide a single scalar value feedback for a single policy learning problem. The scalarization mechanism in this work can be described as [81]: $\hat{SV}_{linear}(s,a) = \sum_{i=1}^{m} w_i \times V_i(s,a)$, where $V_i(s,a)$ is intended to be a value estimator for v_i^{π} and w_i are weights. Now the problem becomes how to achieve multiple value estimators $V_i(s,a)$, one for each value component of our problem.

Based on the refined problem nature, the purpose of estimating multiple real value component outputs via training with the same training data can be modeled as a multi-target learning (regression) problem. Accordingly, we decide to utilize a partial network sharing skill to potentially facilitate the process. It uses multiple networks sharing the front a few layers, where one network aims for one of the expected value component estimators. Here, multi-target learning focuses on simultaneously training networks as value components estimators desired by the formulated scalarized single-policy MORL algorithm with potential intertwine, where the shared layers serve as a joint structure aiming to capture abstract input encoding shareable and beneficial to all networks. This model construction and learning mechanism enable the value estimation at its individual component level.

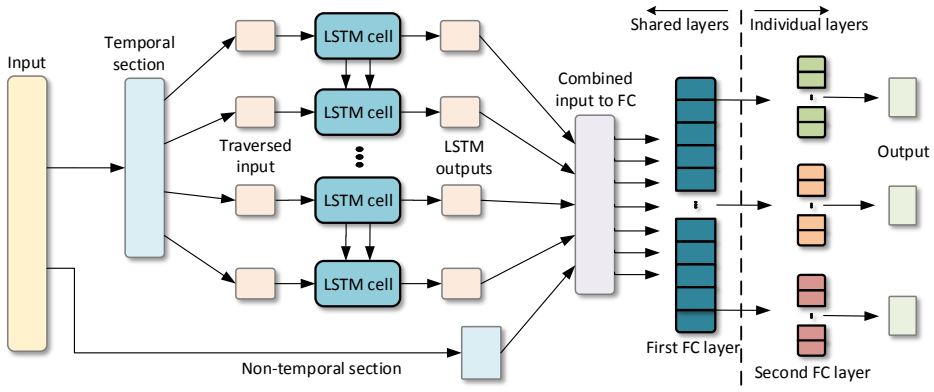


Figure 5.3: The structure of our deep neural network.

The overall network structure of this work is depicted in Figure 5.3. As illustrated, the input is separated into temporal related and non-temporal related sections. The temporal section will first go through the LSTM-based structure and the aggregated temporal hidden states will be integrated with the non-temporal section in the original input to enter the fully-connected (FC) layers. The LSTM session network and the first FC layer are the shared layers, whereas the second FC layer is isolated for each estimator, three in total, corresponding to each of the following value components:

$$v_1 = -\frac{\eta_c \cdot m_{ih} \cdot m_{ic}}{\eta_j} \left(M_j + \sum_{t=t_s}^{t_e} \beta^{D_t} W_j^{(t)} \right)$$

$$v_2 = -\frac{\eta_c \cdot m_{ih} \cdot m_{ic}}{\eta_j} \left(\sum_{t=t_s}^{t_e} \beta^{D_t} W_{cl}^{(t)} \right)$$

$$v_3 = -\psi_{ih} \cdot \psi_{ic} \cdot R_j$$

Here v_1 focuses on feedback related to missing deadline events of job j itself, v_2 focuses on mutual influence of missing deadline events of all jobs on-board the cluster, and v_3 focuses on average execution delay ratio of job j. The integrated value of an action is then decided based on the (weighted) sum of outputs of the three network estimators.

Training Enhancement Skills

We apply several training enhancement skills that are suitable for the model training of the underlying problem with the potential to increase training efficiency and performance. They are:

Cluster occupation status traverse:

The cluster occupation status in the last 105 time steps is specially traversed to generate the actual input of the LSTM session of the model, which also forms the temporal portion of the model input. This can be seen as a segmentation process of the original status vector, aiming at transforming it into a series of status segments (possibly with overlaps) following the same temporal order, each as a continual sub-sequence of the original one. Such kind of segmentation helps preserve the original temporal series information to be captured by LSTM structure, while revealing occupation evolution by the sub-sequence in each segment available to the LSTM cell.

For each cluster, as depicted in Figure 5.4, the process starts at the beginning of the occupation status vector with a 10-element increment for each segment's start-point and with length 15. Therefore, each two consecutive segments have a 5-element overlap. Overall, the segmentation for the length 105 status vector results in 10 segments of length 15, concatenated as a 150 length vector. For totally 5 clusters as in the experiment, we have a 750 length vector as the actual temporal portion of input.

Training with decayed learning rate: We utilize Adam optimization [37] for learning rate tuning. Here we denote it as $Adam(\alpha)$, where α is a base learning rate. Our exemplary experiments reveal that a relatively large base learning rate, such as $Adam(1e-2)$ or $Adam(1e-3)$ may adversely affect training performance. And a single base learning rate with a smaller α shows relatively promising influence, such as $Adam(1e-4)$. However, for these approaches, no manual decay in the base learning rate is included.

We decide to further supplement a manually decayed base learning rate to Adam. More specifically, for the first 1000 training episodes in the experiment, the learning rate is $Adam(1e-4)$, and afterwards, it becomes $Adam(1e-5)$. Such Adam optimization with a manually decayed base learning rate provides the potential to combine the benefit of more swift change in the early stage and more fine tuning in the later stage of the training.

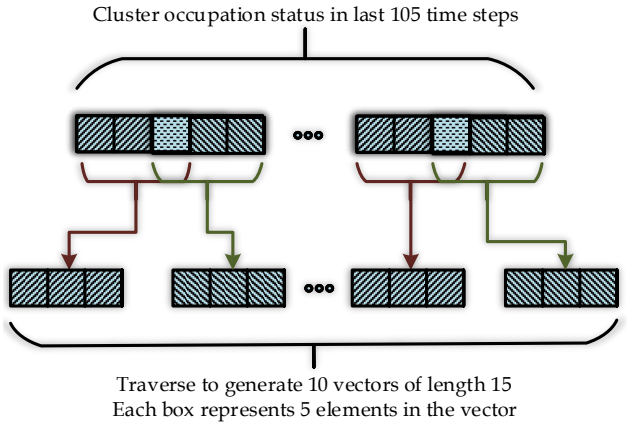


Figure 5.4: Traverse of cluster occupation status.

We've also adopted some training skills in [51], shown to be effective for training RL-based resource management, including:

Training with randomized workload: Randomness is added to important job feature variables to inject variations of the workload in each episode. This stimulates better state space exploration and pressurizes the network in continually refining itself towards generalized knowledge of its duty to suit workload variations.

Modified ε-greedy exploration: A rule-based baseline model is supplemented to guide the action perturbation for exploration. When a random action perturbation is needed (by probability ε from the original ε-greedy exploration), it then has another probability that this perturbation will be instead provided directly by the supplemented baseline, otherwise, a normal random action perturbation is

performed. This is in purpose of letting a rule-based model inject its better-than-random knowledge to partially guide the general exploration, such that exploration efficiency is increased.

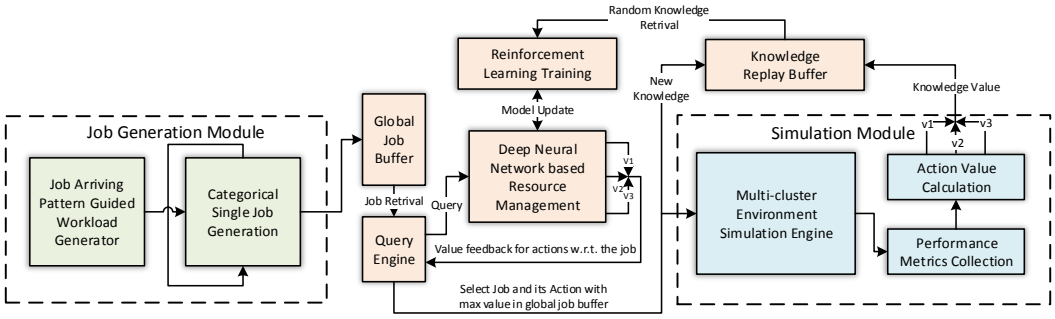


Figure 5.5: Training architecture of our deep neural network in one episode.

Solving multi-job selection dilemma: The proposed resource management can schedule one job at a moment. So, besides deciding the scheduling action for a job, it also needs to supplementarily select a targeted job for its scheduling, if more than one exist in the global buffer. To accomplish this goal, designing a model to accept all jobs in the global buffer at once is difficult due to the absence of job count upper bound and the further exacerbated exploration difficulty. Therefore in the adopted design, our model still accepts a single job input at once, but an outer level iterator will traverse the global job buffer where an overall value vector would be assembled to provide job and its action selection in one integration.

The overall training architecture is shown in Figure 5.5.

Experiments

The experiments are conducted via simulation. In the experiments, the testing distributed computing environment is set as composed of 5 clusters, with executor capabilities for each described by the vector: $[800, 1200, 1500, 1800, 2300]$. Notice that here executors among different clusters can be

heterogeneous. Accordingly, the heterogeneity factor for each cluster is: $[70, 60, 100, 80, 90]$. For elasticity, we suppose two of the clusters: the 3rd cluster and the 5th cluster have elasticity and their resource variation behaviors are managed by the elasticity controller as described below.

Elasticity controller: Executor capacity of an elastic cluster can be expanded by an increment amount (100), when the cluster occupation ratio in any moment of a past time window (100 steps) is $\geq 95\%$. The accumulative expansion in effect can be no larger than a maximum limit (200). On the other hand, if no occupation $\geq 95\%$ is detected in the past time window, the expansion capacity in effect will be returned in stages by a decrement amount (100).

Note that our approach focuses on cluster-level global resource management. To accomplish job execution, a local intra-cluster scheduler is still needed. Our approach, instead of replacing or intervening, will actually cooperate with the local scheduler and as a benefit has the potential to coordinate with different local schedulers. We use a popular generic local scheduler for experiment.

Local intra-cluster scheduler: At each moment in a local cluster, if there are jobs halting and awaiting for resource from previous moments, they are prioritized for resource satisfaction. After that, job requests to start execution at this moment are considered for resource allocation. Whenever queuing is needed, the same sequence as the job arrival is followed. If a job's resource need cannot be satisfied, it will be put into halting (resource waiting), and try again in the next moment. Here, resource satisfaction means that the job's executor request can be fully fulfilled by the cluster.

To compare with our deep RL-based approach, multiple rule-based baselines are considered:

Random (RAN): Jobs are deployed to one of the clusters forming the multi-cluster computing environment solely by randomness.

Round-Robin (RR): Jobs are deployed to each cluster forming the multi-cluster computing environment in a round-robin manner, starting from a random one at the very beginning.

Most Available First (MAF): Cluster with averagely most available computing capacity in a past time window relatively for the considered job will be selected for job deployment. Here the available computing capability for cluster i is defined as: $(1-RO_i)\times CAP_i/ER_i$, where RO_i, CAP_i and ER_i are the average occupation ratio of cluster i in the past time window, current total capacity of cluster i (elastic capacity in effect is included) and the most likely executor amount request from the job to cluster i, respectively.

For all baselines, the job with the least amount of worker request in the global job buffer will be selected. Similar as before, the worker request of the job is defined as the average of the number of executors most likely to be requested by the job with respect to each cluster. In this way, the MAF baseline roughly mimics the idea of 'SF-E', which is the best baseline as presented in [51].

Additionally, we also demonstrate our DRL approach by comparing to **another RL model (denoted as RL-FC)**, which roughly follows the reinforcement learning approach in [51].

From the computing environment perspective, the comprehensive job submission (arriving) pattern to the infrastructure could affect system status changing and thus be relevant to system state transition. In experiments, we use three different stochastic job arriving patterns that are utilized in [51] with the name Uniform, Bernoulli and Beta for thorough approach testing. However, different from [51], where models are trained separately for each job arriving pattern, in this work, the intended model is solely trained with the Uniform job arriving pattern, and its generality of direct usage on other job arriving patterns will be presented. The possibility of having i as the time interval between job arriving events for Uniform pattern is: $P(i)=\frac{1}{|b-a+1|}I_{i\in[a,b]}(i)$ with parameter setting $a=1$ and $b=33$. I is the indicator function. We refer readers to [51] for other supplemental information about job arriving patterns and according formulas for Bernoulli and Beta patterns.

Definition of performance metrics:

Recall there are two major goals for our resource management: (1) reducing number of missing deadline events during job executions, and (2) maintaining low job execution time ratio. These are equivalent as simultaneously minimizing two metrics:

TMDL: Total number of occurrences of missing deadlines for all jobs in all clusters during the execution of the workload.

AJER: Average job execution time ratio among all clusters, *i.e.,*

$$AJER = \sum_{j=1}^{N_w} \left(\frac{100 \cdot \sum_{k=1}^{n_j} \left(\frac{AR_{jk}}{SR} \right)}{n_j \cdot N_w} \right)$$

where SR and AR_{jk} are the standard execution time of job j and the execution time of job j in its k-th running when executing in the designated cluster, respectively. n_j is the number of executions for job j and is usually larger than 1 for Cate-1 jobs. N_w is the number of total jobs in each episode. Note that by dividing AR_{jk} (relevant to cluster heterogeneity) with SR (irrelevant to cluster heterogeneity), the resulting AJER is influenced by heterogeneity of the selected cluster, thereby including the cluster heterogeneity influence into evaluation. For cluster elasticity, its influence is also included in both TMDL and AJER implicitly.

Further, an integrated score $\boldsymbol{S}_{log}$ is defined by caring for the comprehensive performance in both goals.

$$S_{log} = sign(S) * \log_{10}(\max(|S|, 1)) \text{ as } S = -\text{TMDL} + 50 * (100 - \text{AJER})$$

For $S_{log} \in \mathbb{R}$, the higher the better. And a score 0 for S_{log} can be roughly seen as equivalent to having an average execution time same as the standard one and with no missing deadline events. The weighting factor 50 in S is for balancing between two goals, since TMDL is an accumulative measure for all jobs in the workload, whereas AJER is an averaged measure.

To facilitate qualitative comparison, three more comparative measures are defined: Fully-dominant (F), Semi-dominant (S) and Non-dominant (N), which mean that in a testing episode, whether the RL approach has better performance comparing to MAF, for both, only one, or none of the performance metrics (TMDL and AJER in consideration), respectively. For 50 testing episodes we used for approach comparison, the **F/S/N** scores are given as a distribution among all testing episodes. Therefore, for performance preference, F>S>N. Before discussing the experiment results, some training parameters are provided in Table 5.2.

Table 5.2: Some training parameters

Notation	**Description**	**Value**
E	Number of training episodes	1500
N_w	Number of jobs in each episode	1000
K	Capacity of knowledge buffer	50000
B	Training batch size	2000
L_I	Model input layer size	839
L_C	Input size to a LSTM cell	15
N_{fc1}	Neurons in first FC layer	800
N_{fc2}	Neurons in second FC layer (identical size for three networks)	200
L_O	Output layer size (identical size for three networks)	5

Approach Performance:

Consequently, the performance comparison (in S_{log}) for our approach (notated as RL-LSFC) with respect to other baseline approaches RAN, RR, and MAF, in different training episodes during the training process, is shown in Figure 5.6. For each vertical comparison, all models are included to compare for 50 testing episodes. The same workload is used for all models in one testing episode, but is re-generated for each testing episode.

From it, we can observe that among baselines, Random (RAN) and Round-robin (RR) provide similar performance and are significantly surpassed by MAF. For our RL-LSFC, it gradually improves itself during training and surpasses all baselines during the mid of overall episodes,

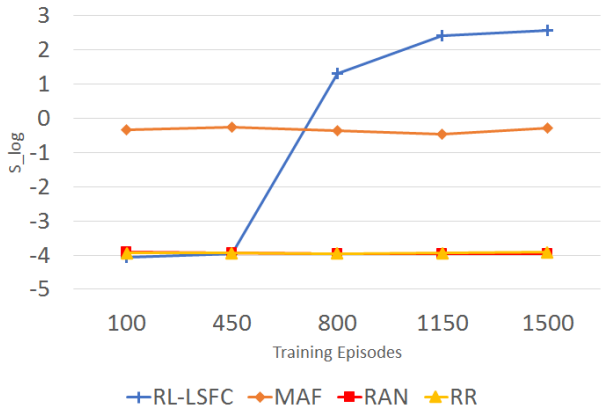
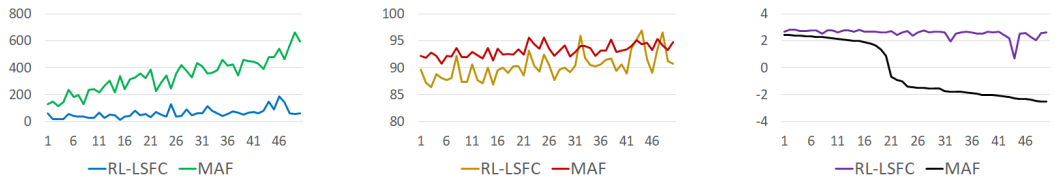


Figure 5.6: Performance comparison (S_{log}) of our RL approach RL-LSFC and baseline approaches in different training episodes.

and continues to increase its performance towards the end of training. A detailed performance comparison between the final RL model after training and MAF is shown in Figure 5.7 and the average statistics are shown in Table 5.3.



(a) TMDL (L) (b) AJER (L) (c) S_{log} (H)

Figure 5.7: Comparison of RL-LSFC and MAF for 50 testing episodes. Three sub-figures are w.r.t. TMDL, AJER and S_{log}. Data in all figures are sorted uniformly in descendent by S_{log} of MAF for viewing convenience. (L) Lower is better. (H) Higher is better.

We can see that our approach trained via deep reinforcement learning outperforms MAF in a large scale. For TMDL, it reduces the occurrence of missing deadline events by 5.57 times. For AJER, it shortens average job execution ratio by 2.92%, thus obtains a significant higher S_{log} score with 2.57 comparing to -0.28. As well, RL-LSFC achieves excellent 46/50 Fully-dominant, 4/50

Table 5.3: Statistics w.r.t. Figure 5.7

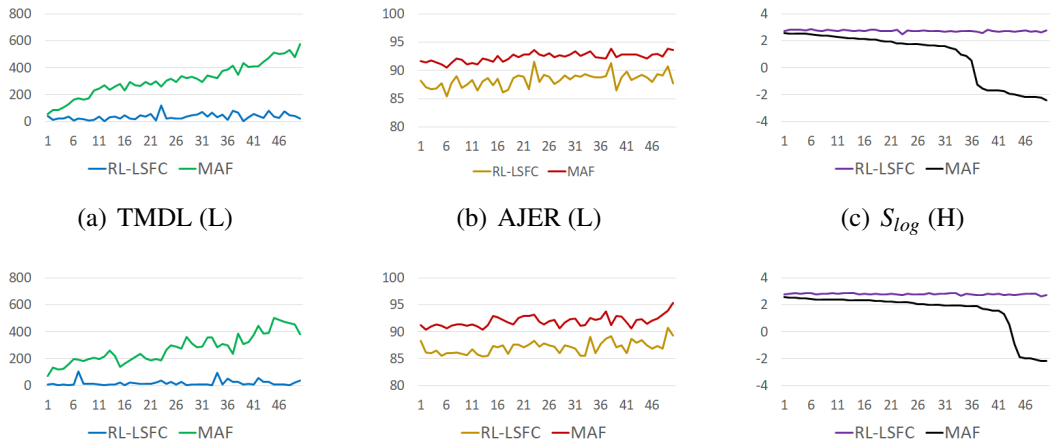
	TMDL (L)	AJER (L)	S_{log} (H)	F/S/N
RL-LSFC	**61.70**	**90.30**	**2.57**	46/4/0
MAF	343.84	93.22	-0.28	-

Semi-dominant and no Non-dominant in 50 testing episodes.

In fact, during experiment, a phenomenon is observed that the model obtained by our deep RL approach can perform equivalently well or even better for workloads that is statistically "less stressful" than the ones in training. Here, a coarse definition of the "stress" can be described as how crowded the computing environment is during the peak period of the workload execution. The workload "stress" can be changed via the Uniform pattern parameter b. In training, we intentionally select $b = 33$, which could reasonably stress the computing environment nearing to its maximum, yet not over-saturate the overall computing capability. And the obtained RL-LSFC could consequently be utilized in a wide-variety of workload scenarios that is "less stressful" than training to the computing environment. Therefore, although RL-LSFC has already shown promising performance in $b = 33$ scenario, we are more caring for its performances in other "less stressful" scenarios, as they represent a much broader variety of applicable cases. We test such generality by two testing scenarios, one with a stress-reduced workload ($b = 36$), and the other with a more stress-reduced workload ($b = 40$). The results are shown in Figure 5.8 and Table 5.4.

Table 5.4: Statistics w.r.t. Figure 5.8

(a)-(c)	TMDL (L)	AJER (L)	S_{log} (H)	F/S/N
RL-LSFC	**37.66**	**88.32**	**2.73**	50/0/0
MAF	311.44	92.35	0.87	-
(d)-(f)	TMDL (L)	AJER (L)	S_{log} (H)	F/S/N
RL-LSFC	**19.76**	**87.12**	**2.79**	50/0/0
MAF	276.1	91.95	1.55	-



(a) TMDL (L) (b) AJER (L) (c) S_{log} (H)

(d) TMDL (L) (e) AJER (L) (f) S_{log} (H)

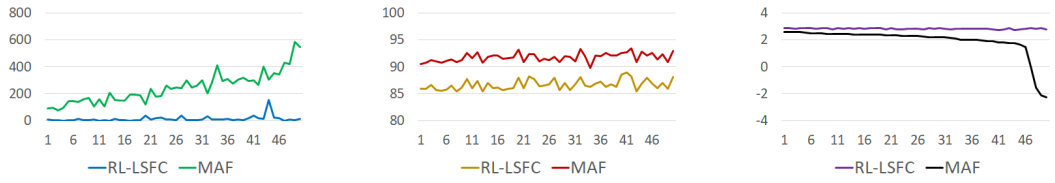
Figure 5.8: Comparison of RL-LSFC and MAF in variant workloads. (a)-(c) are related to $b = 36$ scenario. (d)-(f) are related to $b = 40$ scenario. Other instructions are the same as Figure 5.7.

We can see that as the workload gradually becomes less stressful, RL-LSFC performs even better, and is consistently fully-dominant with respect to the MAF baseline in all testing episodes for both scenarios. Since the stress of the computing environment could largely vary in daily usage, the generality of the RL model to fit for such variation is certainly an apparent advantage.

Performance in other job arriving patterns

Different from [51], where a RL model is trained for each job arriving pattern, we solely train one deep RL model based on the Uniform pattern, and present that the obtained model could work equivalently well for other job arriving patterns directly, which is an out-of-intuition but exciting result. To accomplish, we directly utilize the obtained RL-LSFC model with Uniform pattern onto other two patterns, Bernoulli and Beta. The results in Figure 5.9 and Table 5.5 show that RL-LSFC with respect to the Uniform pattern indeed can work well directly with the other two job arriving patterns. We envision that this may because the obtained model is capable to capture intrinsic information from the provided system status for deciding scheduling actions, and is less prone

and less sensitive to job arriving pattern shift. The observed irrelevancy of job arriving patterns in experiments is a great supplement towards approach generality.



(a) TMDL (L) (b) AJER (L) (c) S_{log} (H)

(d) TMDL (L) (e) AJER (L) (f) S_{log} (H)

Figure 5.9: Comparison of obtained RL-LSFC and MAF in other job arriving patterns. (a)-(c): Bernoulli pattern. (d)-(f): Beta pattern. Other instructions are the same as Figure 5.7.

Table 5.5: Statistics w.r.t. Figure 5.9

(a)-(c)	TMDL (L)	AJER (L)	S_{log} (H)	F/S/N
RL-LSFC	**13.32**	**86.67**	**2.81**	50/0/0
MAF	243.18	91.75	1.92	-
(d)-(f)	TMDL (L)	AJER (L)	S_{log} (H)	F/S/N
RL-LSFC	**39.14**	**88.68**	**2.71**	50/0/0
MAF	295.66	92.41	1.11	-

Comparison with a fully-connected layers model RL-FC

In this experiment, we would like to compare models of our approach to a RL model with fully-connected layers based structure, which roughly follows the ideas in [51] (denoted here as RL-FC).

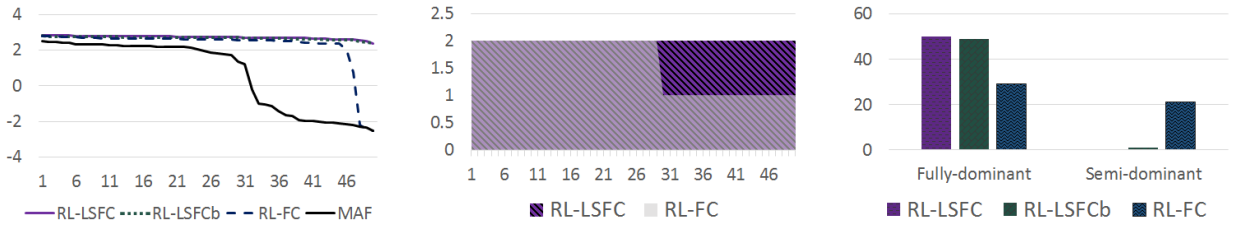
To strengthen the comparison, besides RL-LSFC, we intend to additionally supplement another deep RL model variant also obtained by our approach. Actually, by the greatly separated recognition of

individual objectives of our MORL problem empowered by the approach architecture, we are able to achieve variants of RL models with different balancing between objectives such as via weight adjustment of objectives forming the scalarized value feedback signal. With such benefit, we obtain another variant of our model, namely RL-LSFCb. It shares general conception with RL-LSFC, but with a slightly different balancing between objectives.

Three RL approaches, RL-LSFC, RL-LSFCb and RL-FC together with the MAF baseline are put into 50 testing episodes, results of which are shown in Figure 5.10 and Table 5.6. We can see that although the RL-FC model achieves slightly better TMDL than RL-LSFC, it pays a too large cost in AJER and has thus lower score in S_{log} than both of our models. Similar weakness of it can also be observed in F/S/N distribution comparing to both of our models. Thus, the RL-FC model in experiment shows weaker balancing ability between multiple optimization goals and a weaker overall performance. Meanwhile, two of our models, RL-LSFC and RL-LSFCb show differently prioritized yet better performance. RL-LSFCb could surpass RL-FC in all aspects, which indicates potential advantage of our approach. But its balancing in dual-objectives causes a hit in comprehensive score S_{log} comparing to RL-LSFC and promotes the RL-LSFC model as the best performing model here. The best is, our approach grants the freedom to choose either of them per user preference.

It is worth mentioning that we are not intending to declare an absolute structural or approach advantage here, since a direct and thorough competition of two RL approaches requires extensive hyper-parameter searching and tuning, which is not the main objective and beyond the scope of this work. Nonetheless, the exemplary RL comparative experiments here show that our approach presents decent capability in balancing multi-objectives while providing good overall scheduling performances.

Model behavior pattern exploration



(a) S_{log} (H) (b) Dominant area (c) F/S/N distribution

Figure 5.10: Comparison of three RL models w.r.t. MAF. (a) each curve is independently sorted for viewing convenience. In (b), we give F:2, S:1 and N:0 for scoring to show a dominant area (larger is better) of RL-LSFC and RL-FC (RL-LSFCb is very similar to RL-LSFC here and is omitted for viewing). RL-LSFC indeed has a much larger area (difference as in the pure purple area) than RL-FC. (c) Non-dominant column is omitted since all models have 0 in it.

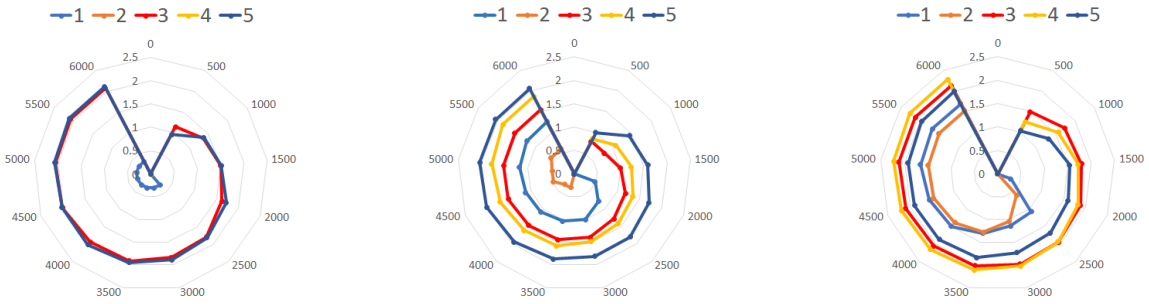
Table 5.6: Statistics w.r.t. Figure 5.10, * are our models

	TMDL (L)	AJER (L)	S_{log} (H)	F/S/N
RL-LSFC*	36.84	**88.71**	**2.71**	**50/0/0**
RL-LSFCb*	**14.74**	90.13	2.67	49/1/0
RL-FC	15.28	92.79	2.24	29/21/0
MAF	305.02	92.67	0.65	-

By observing the good performance of RL-LSFC, we would like to further explore interesting behavior patterns of it. We firstly draw temporal job scheduling sequence in one episode (one point for each job, as shown in Figure 5.11 (a)-(b)) of both RL-LSFC and MAF with respect to all computing clusters, with one color representing one job category. We discover that the deep RL model RL-LSFC indeed shows significantly differentiated scheduling pattern from MAF, which could be coarsely summarized as RL-LSFC tends to utilize clusters with larger capacities and larger heterogeneity factors (thus stronger computing capabilities) more during the early stage of the episode when computing pressure is not peaked, in favor of better performance in both goals. Yet when pressure rises, it well utilizes all cluster resource for overall performance.

Secondly, we further examine the model behavior variance for different job categories (as in Figure

5.11 (c)-(h)). It seems RL-LSFC can successfully distinguish jobs in different categories and provide differentiated scheduling patterns. Such as, for streaming jobs (Cate-1), it seems to prioritize clusters with stronger computing capability and with elasticity, presumably due to their better potentials in suiting streaming jobs with fluctuated executor amount requests. Such differentiated job categorical scheduling patterns are further illustrated in Figure 5.12.

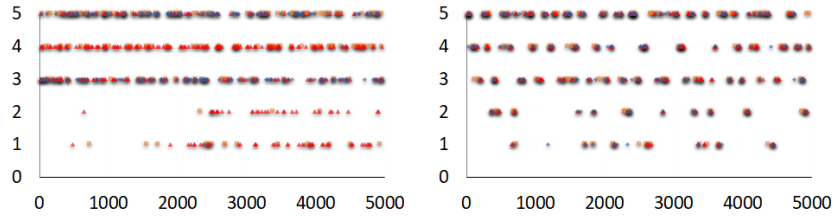


(a) RL-LSFC Cate-1 (b) RL-LSFC Cate-2 (c) RL-LSFC Cate-3

Figure 5.12: Comparison of Job-Cluster scheduling pattern with respect to different job categories under RL-LSFC control. Value axis is on logarithmic scale of job counts, angle axis is time slice. One color for each cluster.

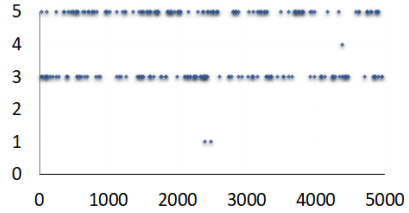
Summary

In this chapter, we present an elasticity-compatible resource management approach obtained via DRL for a heterogeneous multi-cluster computing environment. In experiment, comparing to the best baseline, it successfully reduces the occurrence of missing execution deadline events for workloads of 1000 jobs by around 5x to 18x in different scenarios and reduces average execution time ratio by around 2% to 5%; It also shows better performance than a previous RL based approach with fully-connected layers. We believe this work can contribute to the progress of utilizing DRL in tackling problems related to large-scale distributed computing environments.

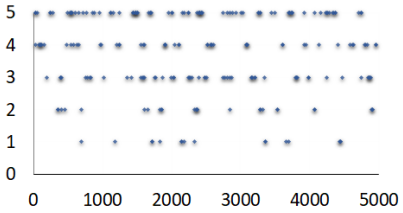


(a) RL-LSFC overall

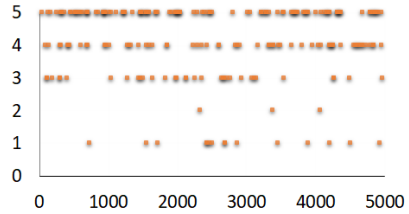
(b) MAF overall



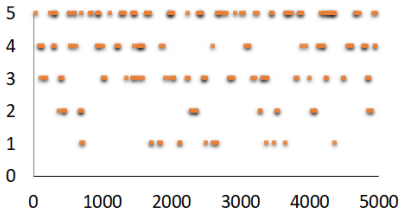
(c) RL-LSFC Cate-1



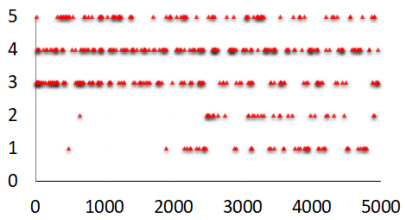
(d) MAF Cate-1



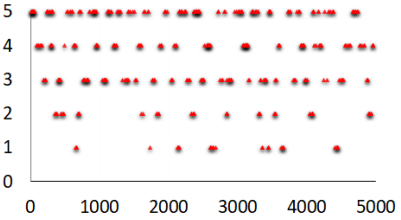
(e) RL-LSFC Cate-2



(f) MAF Cate-2



(g) RL-LSFC Cate-3



(h) MAF Cate-3

Figure 5.11: Job-Cluster scheduling patterns for RL-LSFC and MAF in one testing episode. One point for each job and one color for each category. Vertical axis 1-5 are referring to cluster sequence number. Horizontal axis is time slice.

CHAPTER 6: HYBRID ASYNCHRONOUS APPROACH TOWARDS EFFICIENT PRIVACY-PRESERVING FEDERATED LEARNING

Federated learning (FL) [38] provides collaborative learning for multiple participants with attention on aspects such as data privacy protection, large scale of participants, communication conservation and data heterogeneity, where privacy is one of its foremost considerations [60, 6, 27]. Similar to other forms of machine learning, FL faces challenges from privacy-targeted attacks [32, 44, 45]. The privacy information involved in FL can be majorly classified into three categories: information of a record, information of a set of records (or class of records) and information attributed to a specific participant.

Attacking and defending related to the first two categories are shared by many forms of machine learning (ML) thus are widely studied and non-unique to FL. For example, membership inference attack [72][61] attempts to induce whether a specific sample is included in the training set and model inversion attack [101] aims at speculating properties related to the training dataset. Measures such as record-level Differential Privacy (DP) are shown to be effective against these attacks [68]. These measures could be applied to FL as well. On the contrary, privacy related to participant-level information is more specific to distributed learning such as FL which is less studied. Our work is consequently concentrated on participant-level privacy protection, a characteristic and key privacy protection aspect for FL. To defend against such kind of attacks, specific defending methods need to be designed due to the more abstracted granularity. Participant-level DP is an option against such attacks. However, it intrinsically requires a large scale of participants to converge [24]. Since it perturbs participant-level information instead of hiding them, a balance between model performance and privacy protection is needed. Another option is Homomorphic Encryption (HE), which could compute directly on ciphertext with respect to certain arithmetic operations. However, HE is highly

computing-extensive and needs polynomial approximation to non-linear functions commonly used in contemporary ML, again causing a trade-off between privacy protection and accuracy [93].

Participant-level private information during FL is in fact side-channel information that should be hidden. With such an awareness, Secure Multi-party Computation (SMC) is employed for hiding participant privacy during FL [8]. It is effective by securely merging information from multiple participants without leaking individual information. Additionally, unlike DP or HE, it does not require information perturbation nor function approximation which is much more beneficial to model convergence. It also has better availability in many scenarios with no requirement on scale of participants. The advantage of SMC makes it a routine technique for synchronous federated learning (SynFL). However, SynFL suffers from the synchronization burden. Although Asynchronous FL (AsynFL) helps eliminate such overhead [13, 53, 91], at a cost, the uncontrolled asynchronicity greatly hinders the chance of directly applying SMC due to its decoupled nature. With the indispensable advantages of SMC as stated, revising SMC techniques and making SMC-alike technique available under AsynFL is becoming essential and critical. “-alike" here is referring to the concept of having to utilize a different approach yet eventually accomplish an equivalent privacy protection effect.

In this regard, we propose HALE-Fed, a **H**ybrid **A**synchronous **L**earning towards **E**fficient privacy-preserving **Fed**rated learning framework. We optimize the FL server-participant structure by supplementing a broker layer in between the server and participants, and enable our SMC-alike technique along with asynchronicity. “Hybrid” here means the system runs asynchronously except the broker layer, yet by our design, the system achieves near pure-asynchronous efficiency, which is discussed in more details in Section 6. Furthermore, based on the HALE-Fed framework, we propose an algorithm, namely Fed-SUDA, for solving practical model drifting concern for AsynFL in heterogeneous environments. Experiments demonstrate the efficacy of both HALE-Fed and Fed-SUDA.

The HALE-Fed Framework

Before introducing the architecture of HALE-Fed, let us first briefly recall the architecture of original federated learning. It contains two major components (layers): the server and the participants. During training process, participants obtain the latest model from the server and computes local updates, and then send the updates information back to the server where the aggregated update information is used to apply a model training round for model improvement. For synchronous FL, participants join a model training round in groups and the server awaits and utilizes the aggregated update information from the group. For asynchronous FL, each participant compute and send update to server individually with their own pace, and the server applies model training gradually when sufficient participant updates are received.

As previously mentioned, it is acknowledged that asynchronous organization benefits FL efficiency. In privacy perspective, when system rules are properly configured, asynchronicity brings no extra noteworthy privacy threat except that it hinders the appliance of SMC technique which is a strong and foremost routine privacy protection measure. The obstruction is because vanilla asynchronous FL hinders SMC as all participants operate individually and have no group-based mechanism. It thus becomes a challenge but with significant rewards if one could combine the benefits of both asynchronicity and SMC technique together. Consequently, we propose HALE-Fed that enables SMC-alike technique along with asynchronicity in FL.

Architecture

As shown in Figure 6.1, HALE-Fed preserves the server and participant layers as in ordinary FL, with only modifications to their operation procedures. The foremost innovation is the appending of the broker layer. We require that the broker layer is configured with multiple (more than one) brokers

with similar reliability as the server. The peak computing and storage pressure in each broker by our design is at most similar if not less than the traditional server role in FL, thus is totally realizable. Also, having two brokers totally fulfills the desire (including privacy protection purpose), having more than two brokers could improve protection in rare but more rigorous situations, however, could also increase overhead. We elaborate this more in Section 6. The main components of HALE-Fed thus are:

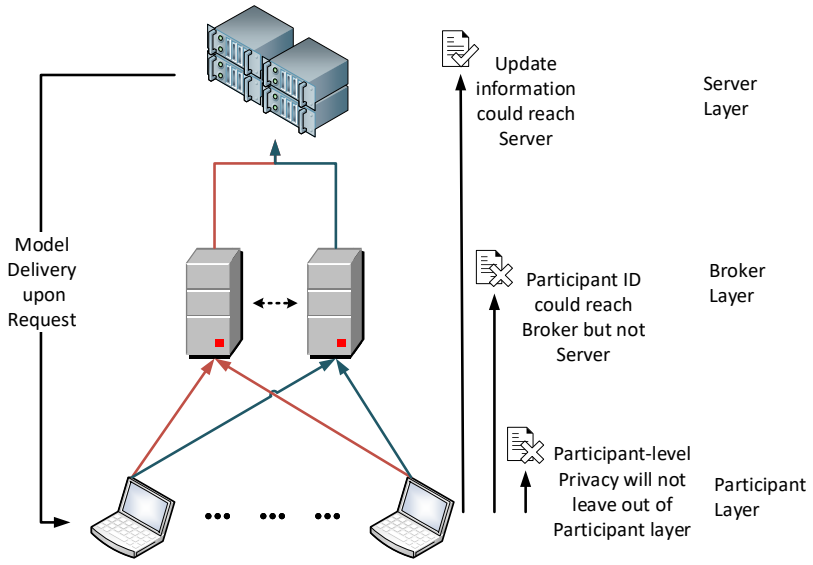


Figure 6.1: Architecture of HALE-Fed.

- **Server** conducts model updating and provides the latest model to participants when being requested.
- **Participants** compute local update based on the latest model from server, split the update into shards, and send each shard to the corresponding broker.
- **Brokers** manage update shards from participants, coordinate with other brokers for forming update

components with each an aggregation of according update shards, and send update components to server.

Participant Update Information Flow

The most significant change brought by our innovation in HALE-Fed architecture is the change in the information flow of participant generated updates and its further induced security and efficiency benefits. To better explain this, we highlight foremost characteristics of the proposed approach as follows and illustrate significant participant update information flow features as in Figure 6.2 :

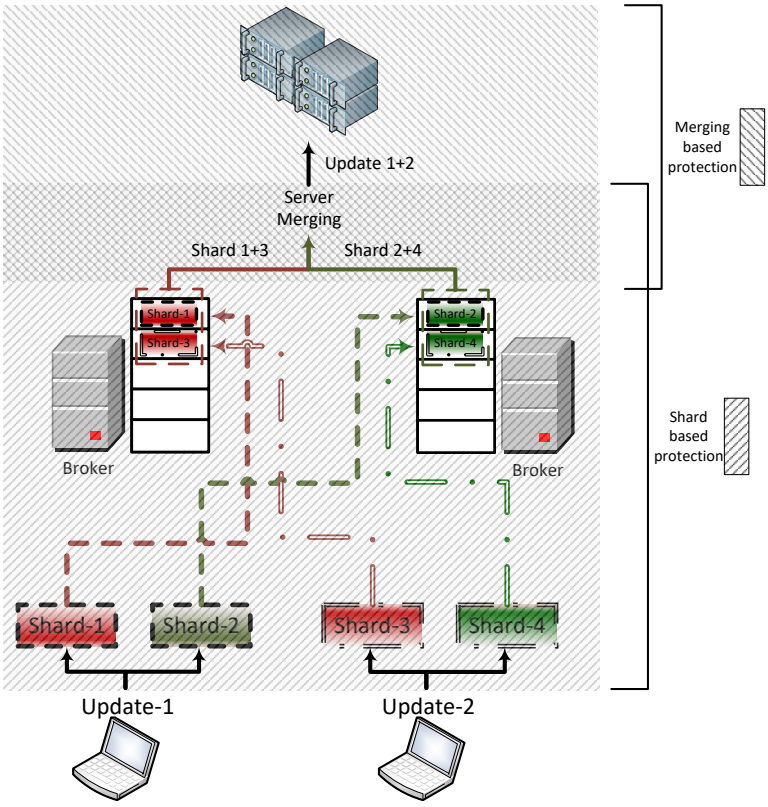


Figure 6.2: Participant update information flow of HALE-Fed.

- In the proposed architecture, all participant generated updates will not be directly sent to server but to all brokers instead.
- Before sending out its update, each participant splits it into shards equalling to the number of brokers, and send each shard to the corresponding broker. Each shard contains only a portion of its update information with mask values (obfuscation) supplemented for extra privacy protection.
- When preparing for a server update round, all brokers confirm a same list of participant generated updates intending to be used, and each broker locally aggregates its owned shards corresponding to the updates in the list and sends to the server. In this way, server receives a update component from each broker in an update round, which is an aggregation of multiple participant generated shards.
- The server will aggregate all components to recover the overall update information without loss. After which, a round of model update is applied and the new model is enabled when updating is finished.
- The participant update information during the entire transmission is at least protected by either shard-based and/or merging-based protection such that private information are not exposed to other system components (other participants, brokers and server).
- Such change in the information flow of participant generated updates has astonishing benefit that it allows most components of the FL system to run in a asynchronous pattern meanwhile providing opportunity for enabling SMC-alike technique. We elaborate this more in the later section.

Shard Transmission

The information transmission during such operation is formally verified by Formula 6.1. Suppose at a certain update round, a set of W participant generated updates $\{D_i\}_{i=1}^{W}$ are used for composing

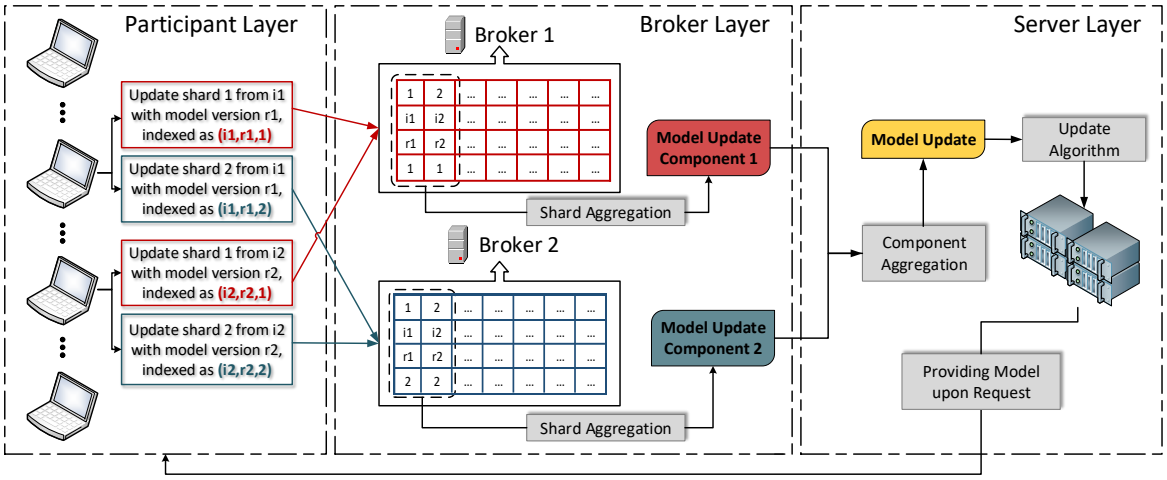


Figure 6.3: Procedure of HALE-Fed.

server model update, thus we should have $U = \sum_{i=1}^{W} \omega_i D_i$, where ω_i is the weight for D_i. $spl_{i,j}$ denotes the j-th split of update D_i, which is equivalent to the corresponding update shard $S_{i,j}$ but without the mask $ma^+_{i,j}$, where $\sum_{j=1}^{B} ma^+_{i,j} = 0$ for $\forall i$ and B is the total number of brokers. C_j stands for the server update component from j-th broker, which is the (weighted) sum of its local update shards for the current round, thus $C_j = \sum_{i=1}^{W} \omega_i S_{i,j}$.

$$
\begin{aligned}
D_i &= \sum_{j=1}^{B} (spl_{i,j}) = \sum_{j=1}^{B} \left(spl_{i,j} + ma^+_{i,j} \right) \\
&= \sum_{j=1}^{B} S_{i,j} \quad 1 \le i \le W \\
U &= \sum_{j=1}^{B} C_j = \sum_{j=1}^{B} \sum_{i=1}^{W} \omega_i S_{i,j} \\
&= \sum_{i=1}^{W} \sum_{j=1}^{B} \omega_i S_{i,j} = \sum_{i=1}^{W} \omega_i \sum_{j=1}^{B} \left(spl_{i,j} + ma^+_{i,j} \right) \\
&= \sum_{i=1}^{W} \omega_i \sum_{j=1}^{B} (spl_{i,j}) = \sum_{i=1}^{W} \omega_i D_i
\end{aligned}
\tag{6.1}
$$

This formula reveals that with the entire transmission, the intended model update U remains unchanged. This leads to the fact that with the supplement of the broker layer, the aggregated update information is transmitted to the server without loss.

System Operation

Based on the previously explained mechanism, we illustrate the system operation procedure of HALE-Fed in a model training round in more details as in Figure 6.3. For a participant, after obtaining the latest model and compute the update, it splits the update into number of shards equaling to the number of brokers with the adopted shard generation algorithm. The information contained in each shard is a combination of partial update information and obfuscation values. Each shard can be marked using the identification tuple (p_i, r_α, γ), where p_i represents the source participant of the shard, r_α the model version number it based on, and γ the broker id it should be sent to. Shards are then sent to the corresponding brokers and stored in broker's shard buffer.

Each broker will receive one corresponding shard from each participant's update. Each broker maintains a shard buffer to store all received shards including its shard identification tuple. The stored shards will remain in the shard buffer until it is successfully used in a server model update or it is decayed for more than the maximum allowed staleness. After either case, the shard will be removed from the buffer. When enough shards are received by the broker layer, it will be responsible for confirming a list of updates that will be used to compose the next server update. Each broker will then compose its server update component (potentially weighted aggregation of according participant update shards) and send to the server.

The server will aggregate all components to recover the overall update information without loss. After which, a round of model update is applied and the new model is enabled when updating is finished. Meanwhile, the server will always provide the currently latest available model to requested participants.

Communication Failure Dealing Mechanism

As a general FL process commonly accepts the participation of many consumer level devices which are connected via diversified networking conditions, it is possible for participants in FL to experience temporary communication failures. Such situation is much more difficult to deal with in traditional FL especially when SMC techniques are applied. Complex recovery mechanism is often desired in such case due to the recovery burden in group based encryption mechanism. We will elaborate this more in a comparison of HALE-Fed with traditional SMC in a later section. On the contrary, such failure is much easier to deal with in HALE-Fed due to its intrinsically decoupled operation and privacy protection mechanism.

The most complicated process during HALE-Fed's system operation which could be influenced by device communication failure is the participant update uploading process. This is because all other communications of processes in HALE-Fed are one-to-one except that when participant uploading their update in terms of shards, which is in overall an one-to-many process (sending shards split from one participant update to all brokers). In practical realization of HALE-Fed, when a communication failure is encountered during a participant update uploading process, either the participant experiences network issue but remains power-on thus could recognize the situation and can simply re-send the failed update component; or the corresponding participant is offline persistently for a certain period that causes a sustained update inconsistency among different brokers with respect to the corresponding participant update (some shards of the update arrive at brokers successfully, while others don't). In such case, this inconsistency could be noticed during update list confirmation among brokers whenever this update is selected such that the incomplete update could be easily discarded without further concern.

Thus it is more convenient for HALE-Fed to deal with abrupt device offline or communication failure, which is an apparent advantage over traditional SMC techniques.

Threat Model and Privacy Preservation

In this work, we consider the threat model as there exists an “honest-but-curious" adversary in the system. It means that the adversary will follow system operational rules (“honest"), but will try to speculate information leaked from normal operations (“curious"). This is a common setting in privacy protection tasks [5, 66, 29] and there is no constraint in this work on where this adversary locates. Although we assume **one** adversary for concise description, the system is capable of handling multiple adversaries. Firstly, the proposed system could tolerate multiple coexisting adversaries without issues if no communication/cooperation among multiple adversaries is allowed by proper practical system design. Secondly, even if communication/cooperation among multiple adversaries becomes possible, the proposed system still could handle up to N adversaries as long as there are at least $N+1$ brokers, so individual update information remains private to non-owners. Therefore, in the worst case, as long as the system is capable of providing a certain number of brokers, it can tolerate adversaries up to the similar scale. In the rest of this chapter, we use the case with one adversary and two brokers for the purpose of providing clearer descriptions and illustrations.

SMC-alike Functionality and Benefits of HALE-Fed

From privacy perspective, HALE-Fed guarantees that only the aggregated participant update results could be seen by authorized system components, which effectively provides **SMC-alike functionality**. In fact, HALE-Fed provides at least equivalent effects in privacy protection comparing to the traditional SMC which is strong and outstanding to participant-level privacy protection. It is also free of any information perturbation or network structure restrictions. HALE-Fed enables such top-notch unique kind of privacy protection measure in asynchronous FL world without any extra hardware environment requirements for server and participants, and provides at least equivalent

privacy protection effect as traditional SMC techniques.

Comparison with traditional SMC technique

The realization of privacy protection in traditional SMC vitally relies on means including: (1) the privacy protection mechanism requires a group based organization; (2) the masking based information obfuscation and recovery are joint efforts among joining participants; (3) mask related information needs to be communicated among participants.

Such requisites naturally deter SMC from native asynchronous learning where participants' attending are disentangled. They also hinder SMC in FL where certain practical situations are probable. For instance, to deal with unpredictable participant disconnections, traditional SMC applies a complex two-layer secret mark with a compromise in disconnection tolerance upper bound. This not only exacerbates the protocol complexity but also forces the abandoning of the entire updating round if number of disconnected participants ever exceeds the tolerance upper bound.

Furthermore, in order to reduce the massive amount of communication among all participants related to mask information exchange, SMC opts for transmitting only the randomization seeds used for a unified mask generator that is in prior consented by all participants. This significantly reduces the mask communication overhead, but restricts the randomization mask to be vector information independent (denoted here as positional masks). Such type of mask could be suitable for obfuscating information in individual positions of the participant update vector but may not be competent for concealing distributional information privacy that could also exist in the vector. The later of which requires a more deliberated designed vector dependent obfuscation and randomization mask (denoted as distributional masks) that could not be communicated merely via the transmission of randomization seed. Therefore when facing with the protection of such distributional privacy in the update vector, SMC encounters a dilemma between privacy protection effect and large

communication overhead.

Different from traditional SMC in FL where mask information needs to be communicated among participants, the mask used in HALE-Fed can be fully decided locally by each corresponding participant, which perfectly coincides with the decoupling nature of asynchronous FL and enables large benefits in mark design freedom, system efficiency and communication failure dealing mechanism. Notice that with the intrinsically completely disentangled nature of masking process in HALE-Fed, it could naturally choose between positional masks or distributional masks for protecting privacy in participant update vector with no apparent difference in cost. Whereas in SMC, if distributional vector privacy is also considered, the corresponding costs can raise significantly.

Most importantly, HALE-Fed provides unparalleled functionality advantage over traditional SMC technique in its extra suitability to asynchronous learning and more convenient failure handling mechanism.

Benefits of HALE-Fed

We now summarize the advantages of HALE-Fed as follows:

- HALE-Fed enables effective SMC-alike technique for participant-level privacy protection in asynchronous FL with no convergence-influencing perturbations, no specific hardware environment requirement in server and participants, and no heavy overhead. It could also natively hide the attending participant list of any round from non-necessary access (the broker layer could harmlessly monitor and keep track of the list, while other system components are restricted from accessing such), which further enhances privacy protection.
- HALE-Fed is more convenient in dealing with abrupt device offline or communication failure and has no masking related mutual communications within system, which also yield advantages over

traditional SMC.

- HALE-Fed can have similar efficiency as pure-asynchronous learning. Recall that all system components in HALE-Fed remain asynchronous except for the broker layer. However, the scale of the broker layer is almost ignorable comparing to the scale of the participant layer, and the insignificant broker layer coordination overhead could be well hidden among simultaneous participant-level activities. In fact, HALE-Fed could achieve near pure-asynchronous learning wall time as verified by experiments.

- HALE-Fed has great compatibility and plenty configurable possibilities to assemble most of the currently available algorithms and policies. As a matter of fact, the broker layer concept of HALE-Fed could also be utilized in SynFL for the purpose of reducing overhead related to traditional cross-participant SMC-related coordination.

Fed-SUDA: An Asynchronous FL Algorithm in Heterogeneous Environments Using HALE-Fed

Previously, we have seen the benefits of our proposed HALE-Fed especially to asynchronous FL. Furthermore as a underlying framework, HALE-Fed enables the possibility to support FL algorithms built upon it. To demonstrate this advantage, we propose an asynchronous FL algorithm named Fed-SUDA that tackles with issues of asynchronous FL with participants having **he**terogeneous **d**ata (Non-IID data) and heterogeneous **c**apabilities (shorted as **HeDC**) which can benefit from HALE-Fed. Here, HeDC is another practical situation that could interfere with the applicability of asynchronous FL.

Challenges Brought by HeDC

FL could often face participants with heterogeneous data and capabilities (HeDC) in practical scenarios (we also generally refer this situation as a heterogeneous environment in later context). The HeDC here refers to a combination of two situations: (1) heterogeneous data and (2) heterogeneous participant computing & communication capability. Data heterogeneity refers to the situation where participants' data distributions are non-IID. Participant heterogeneity means that participants may have different computing and communication capabilities that resulting in different temporal response distribution in FL. Recall the objective function of FL:

$$\min_{x\in\mathbb{R}^d} f(x) = \min_{x\in\mathbb{R}^d} \sum_{n=1}^{N} p_n f_n(x)$$

where $\{p_n\}_{n=1}^N$ are weights and $\{f_n\}_{n=1}^N$ are local objective functions of participants due to non-IID data distributions. N is the total number of participants. For synchronous FL in HeDC, we could easily enforce a uniform participant sampling and adopt $\{p_n\}_{n=1}^N$ weights when assembling according updates from participants. In this case, the influence of HeDC is mainly expressed as the presence of stragglers that largely affects the efficiency. Asynchronous FL helps alleviate this efficiency issue but needs attention to the resulting convergence-related side-effects of HeDC.

In AsynFL, when participants attend FL with their own paces, the differentiated temporal response distribution implicitly enforces a different unknown sampling policy. It can implicitly deviate the intended weights when assembling participant updates thus impair the learning process with heterogeneous data. A type of approach [71] suggests to individually set the number of local iterations "K" (as in FedAvg style algorithms where multiple local iterations are allowed for each update) for different participants to balance local computing time and alleviate this issue. However, this method has disadvantages: (1) To regulate participant heterogeneity by setting different values of "K", the response speed of all are regulated towards the slowest participants, which greatly harms

overall system efficiency; (2) Differentiated network speed could also greatly influence participants' temporal response distribution but could not be easily regulated by setting "K". (3) More severely biased participant response distribution may not be well balanced by such adjustment, since the choice of "K" could also affect and may bring adverse influence to model convergence. Therefore, we are looking for an approach capable of dealing with asynchronous learning in HeDC more generally without the need to entangle with an individualized "K" setting. Ideally, this algorithm should be capable of training model in HeDC with reliable performance and privacy consideration, meanwhile being more temporally efficient than SynFL.

Design of Fed-SUDA

In this regard, we propose an algorithm to tackle the HeDC issue for AsynFL based on the benefits of HALE-Fed. Roughly speaking, we decompose the training process in terms of **participant episodes** and enforce a participant sampling with no replacement in each episode. In other words, a participant attends once and only once in a participant episode with its update (which can be staled). The key idea is to buffer latest updates from some slower participants (stragglers) at the broker layer as **shadow updates** (in the form of shards), and use them whenever slower participants' updates are needed for proceeding participant episodes. In this way, uniform sampling is unconditionally satisfied yet the system training speed is less restricted by response speeds of slower participants.

We assume stragglers exist in system where their response speeds are much slower than the majority of others due to HeDC, and the response time of each participant follows its own distribution with expectation known. We then regulate a heterogeneity factor "L". L helps decide who and how many of the participants shall utilize the "shadow update" mechanism. Suppose the slowest expected response speed (number of responses over unit time) among all participants is normalized to be 1, then for each participant whose relative response speed is $\leq L$, its latest update will always be

stored. We denote the set of such participants as *SP* (“slow participants"), and the set of other participants as *RP* (“regular participants"). In each participant episode, For participants in *RP*, they join each episode on site by their latest computed response. Notice these updates could as well be staled, since when applying a partial participation policy, updates in a participant episode are capable of composing multiple server update rounds, thus some participant updates will become staled during this process. For participants in *SP*, when update of them are needed for finishing a participant episode, we use its buffered latest available update stored at the broker layer (shadow update). Participants in *SP* will gradually renew their buffered updates based on their capability.

In this way, when participant with unit response speed finishes one response, the system is expected to be capable of finishing approximately L participant episodes. Recall in asynchronous learning, we often regulate that any participant update used should not be staled for more than an upper bound “T” (more details in Section 6). Suppose participant updates in each episode are used to compose “S” rounds of service updates, then a participant update will reach the staleness upper bound in “T/S” episodes. Thus, when choosing $L > T/S$, L has no direct impact on system efficiency. The parameter L has the meaning of balancing buffer cost and heterogeneity tolerance, if buffer budget is large, we may set a large L if needed to better tolerate participant heterogeneity. This **shadow updating** technique could cope with different local and global updating schemes. Here we integrate it with an asynchronous analogy of the FedAvg algorithm and denote it as **S**hadow **U**pdate supported **D**ifference **A**veraging descent for asynchronous **Fed**rated learning, shorted as **Fed-SUDA**.

Fed-SUDA is intended to merge the gap between the expected unbiased sampling and the uncontrolled asynchronous nature with conservative buffering. This will be much more difficult without the proposed HALE-Fed, since buffering individual participant update in an ordinary server-participant structure FL system is a significant threat to privacy. [65] proposes to use Trusted Execution Environments (TEEs) [63] for storing asynchronous gradients at the server. However, the TEEs are practically limited in size such that certain algorithm settings will be restricted, and such

method does not support SMC application with asynchronous learning. In HALE-Fed, the buffering of any individual participant update could be safely done at the broker layer, which enlightens the design of Fed-SUDA. Fed-SUDA could be regarded as being storage-conservative as it requires only a portion of updates be stored and this threshold is customizable. Also, the storage is implemented at the broker layer and be consistent with brokers' duties for buffering participant updates, thus no significant extra burden is applied to the system. In short, Fed-SUDA tries to correct asynchronous FL training in HeDC and takes SMC-alike participant-level privacy protection into consideration.

It is worth noting that the FedAvg algorithm [60] for SynFL may not work if we directly adopt its server updating scheme to asynchronous cases. Recall the global server update formula for FedAvg (if consider partial participation) is:

$$M_{r+1} = \frac{1}{\sum_{i \in N_W} p_i} \sum_{i \in N_W} p_i M_{r+1}^{(i)} \tag{6.2}$$

where M_{r+1} is the new $(r+1)$-th version of server model and $M_{r+1}^{(i)}$ $(i \in N_W)$ are the computed local model for participant i for the $(r+1)$-th round. Here N_W is the set of W participants attending current round. In this formula, the new and current versions of server models have only implicit connection carried by the generation of $M_{r+1}^{(i)}$ $(i \in N_W)$, where each $M_{r+1}^{(i)}$ is obtained from several local SGD steps starting from M_r. This updating scheme is suitable in synchronous situation, but instead may cause problems in asynchronous cases. In asynchronous FL, since the participant updates can be staled, i.e. their start-point models may not be the same letting alone be the currently latest model, the implicit connection between consecutive versions of server models can be entirely broken which could impair model convergence.

In Figure 6.4 (blue curve), we demonstrate how a native asynchronous conversion of FedAvg (Asynchronous Model Averaging, briefed as AsynMA) which roughly follows Eqn. 6.2, fails to converge when two sub-groups of overall participants keep isolatedly and alternatively updating the global FL model and forming oscillation and divergence. Consequently, we propose to utilize

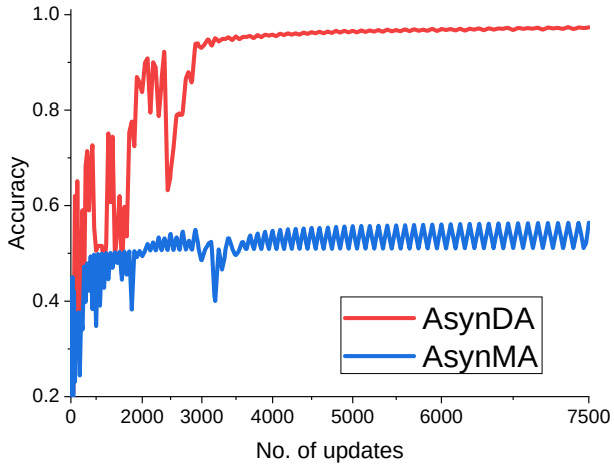


Figure 6.4: Comparison of AsynDA and AsynMA

a server updating scheme for asynchronous FL (Asynchronous Difference Averaging, denoted as AsynDA) that conceptually follows the idea of asynchronous gradient descent, but each increment $D_i = M_{i,K} - M_{i,0}$ is derived from the difference of K-th and 0-th model of each participant in a local K-iteration SGD process. Thus, it could be roughly formulated as:

$$M_{r+1} = M_r + \eta \frac{1}{\sum_{i \in N_W} p_i} \sum_{i \in N_W} p_i D_i \tag{6.3}$$

The benefit of AsynDA is that it constantly enables the connection of consecutive server models during training meanwhile allowing a multi-iteration local participant-level computation. The effect of AsynDA is as well shown in Figure 6.4 (red curve), under the same assumption, it successfully solves the dilemma facing AsynMA and efficiently guides the model towards convergence. We consequently adopt AsynDA scheme in implementation of asynchronous FL algorithms in this work.

Asynchronicity and Staleness

The essence of asynchronous learning comes from the permission of utilizing staled participant updates to eliminate the synchronization barriers that are constraining the training process. The staleness is referring to the fact that the server model used for computing a participant update and the model on which the participant update is used might be different. Quantitatively, the current staleness of a participant generated update could be generally defined as the difference of version numbers of the current global model and the model associated with the generation of the update.

The ingenious intent of Fed-SUDA is to completely transfer the effects of the uncontrolled sampling bias induced by HeDC in asynchronous learning to merely variations in the staleness of the participant updates via the utilization of the shadow updating mechanism. In this way, the crucially disadvantageous influence of HeDC to asynchronous learning is resolved with neither significant sacrifices in the achievable efficiency nor deterioration to synchronicity.

Recall that in asynchronous learning algorithms, it is a general requirement that the staleness of participant updates involved in training be no greater than a upper bound T. This is in order to maintain the desired algorithm convergence property. As previously mentioned, we follow such convention to regulate similar staleness requirement in our algorithm. In this way, our algorithm grants un-biased model updates and regular staleness requirement for asynchronous learning in HeDC, eliminating the influence of HeDC, and consequently make it having analogous properties to asynchronous learning in regular environments. Due to our algorithm design tactic in Fed-SUDA, the staleness regulation not only could help maintaining the desired property of the proposed algorithm, but also could become a suitable and implicit regulation for potential extreme differences in participant heterogeneity. In other words, the system with Fed-SUDA could widely accept the coexistence of different participant heterogeneity and only provisionally regulates the system operation when some participant update staleness exceeds the upper bound limit. This helps to

achieve good asynchronous learning efficiency in a heterogeneous environment.

Algorithm

We now provide algorithmic description of HALE-Fed framework with the realization of Fed-SUDA algorithm, as an example of system procedures for HALE-Fed when coping with algorithms. Nonetheless, the proposed framework is compatible with a wide variety of algorithms solely by procedure modifications. We start with introducing the notations as below.

Notation	**Description**
N, N_{rp}, N_{sp}	Number of total, RP and SP participants
W, W_{rp}, W_{sp}	Number of total, RP and SP updates in a round
γ, η	Participant local learning rate and server learning pace
M_{cur}, r_{cur}	Current model and current version (round number)
$\{C_j^{(r+1)}\}_{j=1}^{B}$	Update component from broker j for round $r+1$

The system repeatedly utilizes the following algorithms until the training is finished:

Participant Layer: (Algorithm 4)

When being able to join, a participant requests and receives the latest server model and then computes the local update based on a multi-iteration local gradient descent process. Before sending out the update, the participant decomposes it into shards equaling to the number of brokers. Then the participant sends out each shard to the corresponding broker, which concludes a typical round of participant duty.

Algorithm 4 HALE-Fed & Fed-SUDA (Participant)

• **Participant** $\{U_i\}_{i=1}^{N}$**:**

∗ **Model request:**

request current server model $M_{lc} = M_{cur}$ and current model version $r_i = r_{cur}$ from server;

∗ **Local update:**

$M_{i,0} = M_{lc}$

for k in range(K): **do**

Sample mini batch $z_{i,k}$ from local data;

Generate new local update by:

$M_{i,k+1} = M_{i,k} - \gamma \nabla f(M_{i,k}; z_{i,k})$;

end for

Compute $D_i = M_{i,K} - M_{i,0}$;

∗ **Send:**

split D_i into B shards $Set_B = \{S_j\}_{j=1}^{B}$ where:

$D_i = \sum_{j=1}^{B} S_j = \sum_{j=1}^{B} (spl_j + ma_j^+)$, and

mask $\{ma_j^+\}_{j=1}^{B}$ satisfies: $\sum_{j=1}^{B} ma_j^+ = \mathbf{0}$;

let $S_{i,j}^{(r_i)} = S_j = spl_j + ma_j^+$ for $j \in [1, \cdots, B]$;

send $S_{i,j}^{(r_i)}$ to broker j;

Algorithm 5 HALE-Fed & Fed-SUDA (Broker)

• **Broker** $\{Br_j\}_{j=1}^{B}$**:**

∗ **Receive shards** $S_{i,j}^{(r_i)}$**:**

store $S_{i,j}^{(r_i)}$ in the buffer;

mark it as from participant i with version stamp r_i;

∗ **Upload update components:**

$W_{rp} = \frac{N_{rp}}{N} W$, $W_{sp} = W - W_{rp}$

if $\geq W_{rp}$ shards from RP users in buffer AND no update in progress **then**

Select one broker as the organizer to do:

· identify current round $r = r_{cur}$;

· select W_{rp} shards from regular buffer and W_{sp} shards from shadow buffer

· ensure the staleness of all shards are $\leq T$ (otherwise, wait for updates to unsatisfied shards), thus forming a list of W shards: $\{S_{i_k,j_k}^{(r_{i_k})}\}_{k=1}^{W}$;

All brokers confirm the list and do:

· $C_j^{(r+1)} = \frac{1}{\sum_{k \in W} p_k} \sum_{k=1}^{W} p_k S_{i_k,j_k}^{(r_{i_k})}$;

· send $C_j^{(r+1)}$ to server;

· when update succeeds, delete the W_{rp} shards used from regular buffer;

Increment participant episode by 1 in every N/W rounds;

end if

Broker Layer: (Algorithm 5**)**

The broker layer coordinates and manages the update shards buffer. When coordinating for an update, one designated broker served as a broker organizer will select a potential list of updates (shards) that will be used for the next round of server update, the list will be confirmed by other brokers. For Fed-SUDA, this could include shadow update shards. The designation of the broker organizer will not affect the framework functionality or benefit the potential honest-but-curious adversary. After that, each broker will generate the aggregation of all shards in the list (as a server update component). Then the components are sent by all brokers to the server.

Algorithm 6 HALE-Fed & Fed-SUDA (Server)

- **Server:**
- **Initialization:**
  if no initial model **then**
    initialize model $M_{cur} = M_0$;
  end if
- **Model:**
  maintain a current model M_{cur};
  if participant requests model **then**
    send M_{cur} and current round number r_{cur} to the participant;
  end if
- **Update:**
  if received update $\{C_j^{(r+1)}\}_{j=1}^{B}$ **then**
    update $M_{r+1} = M_{cur} + \eta \sum_{i=1}^{B} C_j^{(r+1)}$;
    replace current model $M_{cur} = M_{r+1}$;
    send success message to brokers;
  end if

Server Layer: (Algorithm 6)

Server conducts the actual model updating. In each update round, It first coordinates with the brokers to receive all updating components, and after which, aggregates all components to obtain the aggregated information. The server will conduct an atomic model update as specified by the algorithm and enable the new model when the update is entirely completed.

Experiments

The experiments in this work can be categorized into three aspects: (1) Efficacy of HALE-Fed for participant-level privacy protection; (2) Efficiency and performance of HALE-Fed comparing to other organization forms in regular FL. (3) Efficiency and performance of Fed-SUDA (built upon HALE-Fed) comparing to other algorithms in FL with HeDC.

We expand the generality of experiments by varying significant experiment components including: dataset (MNIST[18] and CIFAR-10[39]), participants' non-IID data distributions and heterogeneous/homogeneous response distributions. We also compare variations of our proposed approach with SynFL and AsynFL algorithms. As aforementioned, For SynFL, we use the most popular FedAvg algorithm. For any AsynFL approach, the applied server updating scheme is AsynDA. We could not identify other additional benchmark algorithms, as currently there is no other existing asynchronous algorithms comparable that could both handle HeDC situation and provide SMC-type participant-level privacy protection.

For all accuracy results shown in all tables of experiments, they are averaged over 2 trial runs to alleviate randomness. In each run, we take the mean of testing accuracy in the last 5 training rounds to improve their representativeness. The testing accuracy is obtained by using the 10000 testing images in according dataset. For different approaches, we try to let them share common parameters when reasonable, this is for the purpose of lowering the influence of hyper-parameters to experiments.

Efficacy of SMC-alike Technique in HALE-Fed

We first use Figure 6.5 as an illustration for the necessity of SMC-alike technique in FL by showing what participant-level privacy could be exposed by participant updates. Note in FL, even though

either gradient or model weights could be the update content depending on the employed FL algorithm, they nonetheless are equivalent for privacy perspective since obtaining a certain version of global model in FL is straight forward (especially for the server itself) thus so is the conversion between the two (gradient and model weights).

In Figure 6.5, we visualize part of the update content which are the gradients in the last fully-connected layer (200×10 parameters) of neural networks from different individual participants (with non-IID data) during one exemplary training round with MNIST classification task. Here one sub-figure corresponds to one participant.

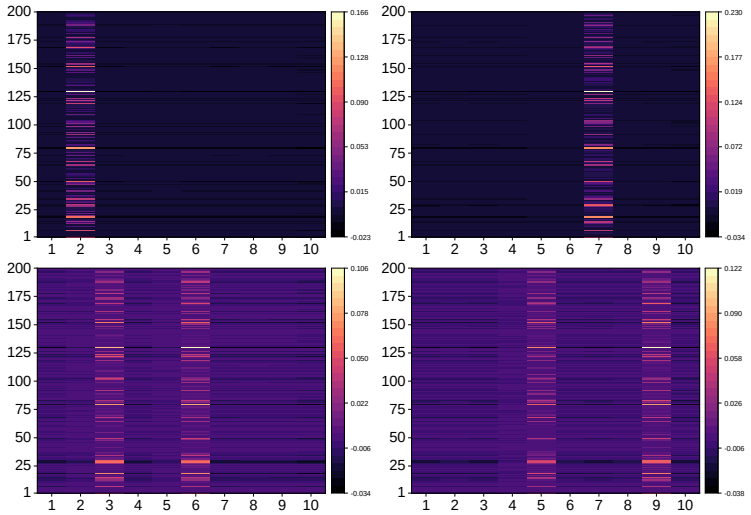


Figure 6.5: An illustrative example of participant-level privacy leakage.

For verification purpose, we let participants in this example possess training data of some specific digits in the MNIST dataset. As a result, we can easily speculate the data distribution of each shown participant by Figure 6.5. For instance, the bottom left sub-figure apparently reveals that the participant dataset mainly contains data of digits "2" (column 3) and "5" (column 6) in MNIST which coincides with the ground truth setting, and other sub-figures follows a similar result. Thus, it is affirmative that individual participant updates could in fact easily expose participant-level

privacy and that participant-level privacy protection measure is indispensably necessary and vital to asynchronous organization of FL where individual response pacing is its key feature.

With the enabling of SMC-alike techniques in HALE-Fed, the adversary could at most see the aggregated update from all currently participating participants in an update round. Such as, suppose we use updates in Figure 6.5 to compose a server update round, the aggregated update is shown in Figure 6.6. From it, the adversary is incapable of accurately speculating any individually specific participant privacy. And the proposed framework also natively hides the list of attending participants of current update round from members having access to the aggregated update, which

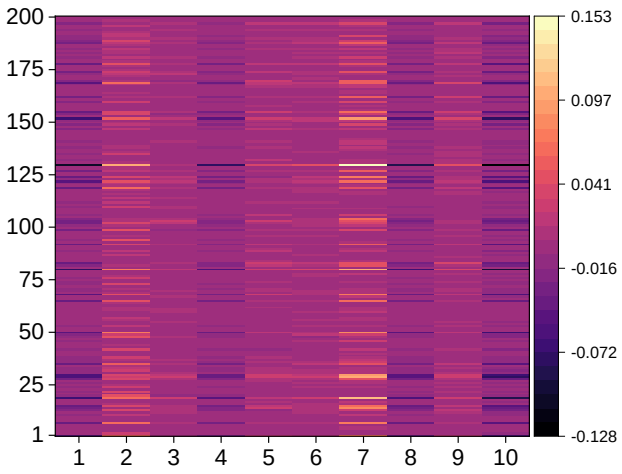


Figure 6.6: Aggregated update result of Figure 6.5 updates.

serves as an additional layer of privacy protection and makes it harder to trace back any leaked participant privacy (if leaking is possible) to the according participant identity. In practice, the number of participant updates used to compose a server update could be significantly larger than the 4 participants as in Figure 6.5, for instance, this number is set as 50 in later experiments. In this way, the aggregated updates becomes more general and privacy-neutral which benefits privacy protection. So being able to see only the aggregated update provides no privilege to the adversary on speculating participant-level privacy.

Next we verify the effectiveness of the proposed shards and brokers related mechanism. Recall that when sending out the update information, participant divides it into shards, and send each shard to a corresponding broker. We demonstrate this process by illustrating the transforming of the gradient content from the same model layer as previously. Specifically, we use the upper right sub-figure of Figure 6.5 as an example and express its transformation in Figure 6.7. With a designed dividing algorithm, it shows that the participant update could be successfully divided into two shards and recover vice versa. We consequently verify that this process could be done smoothly in both directions and the update information is well preserved during transformation.

It is also observable that each shard on the right could successfully hide local data patterns of the update on the left, thus it no only hides individual positional vector information in the participant update but also hides vector distributional privacy. In this way, the participant is safe to send out the shards to according brokers that no broker could speculate its participant-level privacy. Note that the user could customize the update dividing method depending on the characteristics of updates.

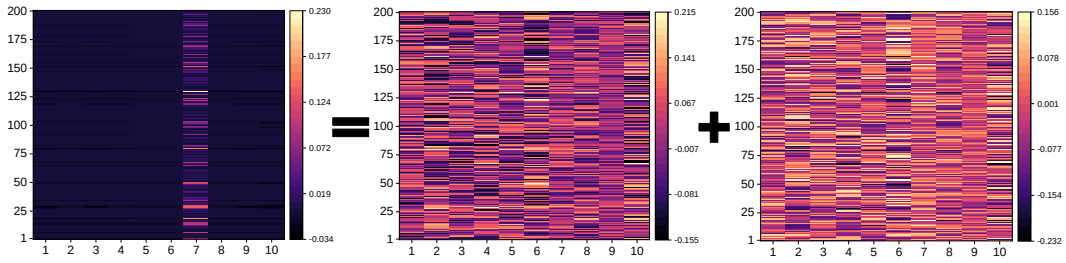


Figure 6.7: Update splitting and merging.

Efficiency and Performance of HALE-Fed

We now lean our attention to verify the efficiency and training performance of HALE-Fed with respect to other organization forms of learning: Synchronous and Asynchronous FL. Recall HALE-Fed can be regarded as a form of (hybrid) asynchronous FL with SMC-alike technique. The other

comparative methods are therefore: SynFL with SMC-alike technique (abbreviated as "Syn"), and vanilla AsynFL without SMC-alike technique (abbreviated as "Asyn"). In this experiment, we train classifiers for MNIST dataset via FL with all these methods. For HALE-Fed, a factor which may influence its performance is the communication overhead between brokers in the broker layer. Consequently, we simulate two variants of HALE-Fed in this experiment, namely the HALE-L and HALE-H, which simulate the performance of HALE-Fed when the communication cost between brokers are **l**ow and **h**igh, correspondingly. Thus, there are in total four candidates, Syn, Asyn, HALE-L and HALE-H.

In this experiment, there are in total 100 participants in the learning process. We divide the 50000 training MNIST images into 200 non-overlapping groups, each with 250 images of one MNIST digit, and each of the 100 participants randomly obtains two groups with no replacement as their local data. Thus the participant data distributions are non-IID.

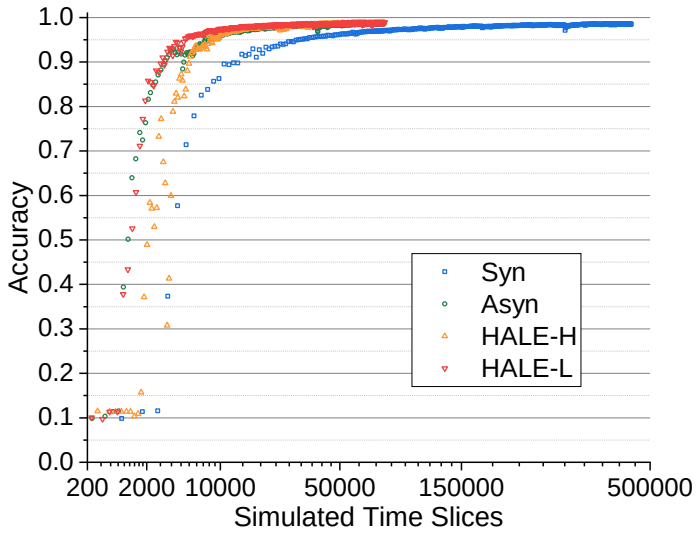


Figure 6.8: Accuracy over wall time.

The results are shown in Figure 6.8 and Table 6.1. From Figure 6.8, it is observable that all four methods provide similar accuracy at the end of training. The huge benefits of non-SynFL approaches

Table 6.1: Training result

Method	Accuracy	Training time
Syn	0.988	45.2×10^4
Asyn	0.987	7.81×10^4
HALE-L	0.988	7.86×10^4
HALE-H	0.987	7.86×10^4

is highlighted when training wall times are considered. As in Table 6.1, SynFL finishes training with the wall time of about 4.5×10^5 time slices, whereas all three non-SynFL approaches finish the training with just about 7.8×10^4, which is a stunning **5.8**x speedup comparing to SynFL. Among the three non-SynFL methods, the training wall times of both HALE-L and HALE-H are almost identical to AsynFL, which shows two important phenomena: (1) HALE-Fed has little overhead comparing to vanilla AsynFL and (2) HALE-Fed is not very sensitive to different coordination overheads in the broker layer. This coincides with our speculation that HALE-Fed could provide nearing to pure asynchronous speed, but additionally supplements the vital SMC-alike participant-level privacy protection to asynchronous FL training which is absent in vanilla AsynFL. Since HALE-L and HALE-H perform reasonably similar to each other, we consider HALE-L condition in later experiments.

Effectiveness of Fed-SUDA built upon HALE-Fed

Next, we focus on demonstrating the effectiveness of Fed-SUDA which is built upon HALE-Fed. In this experiment, the **HeDC** setting is added to the experiment scenario. From Scenario A, we make the following changes: (1) the participant data distribution is re-arranged. We divide the 50000 training images in sequence into 200 groups, and each participant also in sequence takes two groups with no replacement. In this way, the first 10 participants will hold all images of the first digit in MNIST dataset, and so on. The purpose of this is to better observe the influence of HeDC to

the convergence of different approaches. (2) participants now have different response distributions. Specifically, there are 80 regular participants (ID 1-80), where their expected response time could be different but roughly smaller. And 20 slower participants (ID 81-100) , where their expected response time is much longer than the other regular participants (about 20x-50x longer). We also intentionally let all 20 slower participants hold all images of two digits (“8” and “9”) in MNIST dataset, in purpose of isolating their influence for clearer experiment result observation. Here we assume “L” is set so that the 20 slower participants will be “SP” and other participants be “RP”. We refer this setting as experiment **Scenario B**.

Due to features of Fed-SUDA method, the 50 updates for each server update round partially contain updates from RP participants which are never used previously, and shadow updates from SP participants that have been stored and may have been used repeatedly. Therefore to be fair and consistent with other methods, we count the number of participant updates for Fed-SUDA based on uniqueness and avoiding redundant counting for repetitive usage of the same update.

Recall the update staleness upper bound “T" is an important parameter involved in Fed-SUDA. In this experiment, we also generate two variations of Fed-SUDA corresponding to two different requirements on “T" setting. Specifically, Fed-SUDA2 has a 2.5x larger setting value of “T" than Fed-SUDA1 throughout the training process. That is, Fed-SUDA2 allows the participant updates to be more “staled” than Fed-SUDA1. This is for the purpose of testing the influence of “T" to the performance of Fed-SUDA. We as well include two comparative algorithms, SynFL with FedAvg (briefed as “Syn" here, has full control on participant behavior) and AsynFL (briefed as “Asyn" here, has no apparent control on participant behavior). In this experiment, all algorithms are protected by SMC-alike techniques so that the concentration moves on to the proposed Fed-SUDA algorithm in HeDC.

As a result, the accuracy of different approaches over number of updates in heterogeneous envi-

ronment are shown in Figure 6.9. It reveals that both “Syn" and variants of Fed-SUDA achieve reasonable and very similar accuracy at the end of training. However on the contrary, “Asyn" shows much worse training accuracy development and has repeating oscillations which are obvious in its accuracy curve. As stated previously, the problem of AsynFL comes from the combination of its unrestricted decoupled nature and the presenting of HeDC that causes an undesirable convergence drift. As a result, training of AsynFL in this scenario either diverges or converges to an erroneous objective. Therefore, we believe that it is unsafe to directly utilize AsynFL in HeDC situation and argue that a kind of proactive control or correction measure for asynchronous methods (such as those proposed in Fed-SUDA) is essential for obtaining acceptable results meanwhile attending to training efficiency. In fact, we will later use an additional experiment in Section 6 to further demonstrate the incorrectness of AsynFL in HeDC.

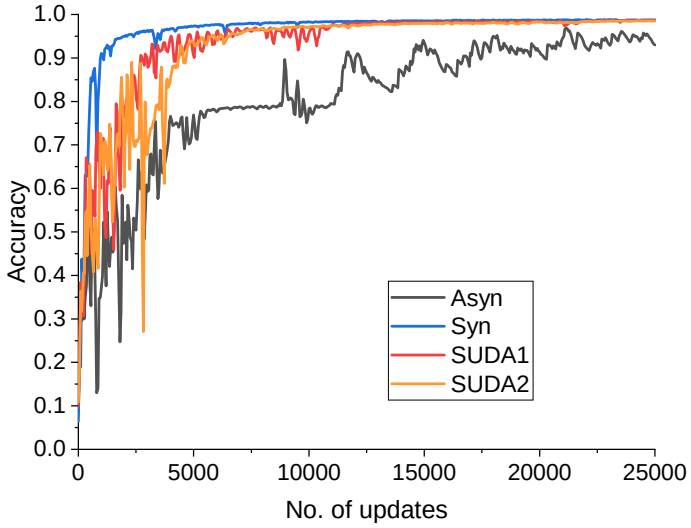


Figure 6.9: Accuracy over number of participant updates.

Due to AsynFL being ruled out in HeDC situation, we now examine the accuracy of other remaining approaches over the wall time which is shown in Figure 6.10 and Table 6.2. They show that both variants of Fed-SUDA have much shorter training wall time than Syn but achieve competitive accuracy. In fact, since Fed-SUDA2 has a more relaxed permission on update staleness, it has less

Table 6.2: Training result

Method	Accuracy	Training time
Syn	0.988	22.54×10^5
Fed-SUDA1	0.987	4.03×10^5
Fed-SUDA2	0.985	1.83×10^5

dependence on the updating frequency of slower participants and presents even less training wall time than Fed-SUDA1. As a result, Fed-SUDA1 and Fed-SUDA2 respectively achieve astounding **5.6**x and **12.3**x speedup over SynFL. This reveals the advantages of Fed-SUDA that it could achieve competitive accuracy and much better training efficiency than SynFL in HeDC. It also has good tolerance on update staleness and most importantly could correct the model convergence drift that causes serious problems in regular AsynFL.

To better assess training efficiency of approaches, we introduce a metric, namely efficiency index (EI), $EI(u) = \frac{a(u)}{t(u)^{0.2}}$, which takes both accuracy and training time into consideration. Here, $a(u)$ and $t(u)$ represent the accuracy and training wall time when utilizing u updates. The exponent power (*e.g.* 0.2) on the denominator is solely in purpose of balancing the scale difference in values of accuracy and training time, which will not change the tendency and comparative relations among different EI curves. As expected, both variants of Fed-SUDA have better efficiency than SynFL with Fed-SUDA2 possessing the best efficiency due to the least training wall time. Note that the EI curve normally gradually decreases since accuracy improvements in later periods of training tend to be less prominent as the model approaches the objective, when comparing to the consistent steady increase of training wall time throughout training.

Heterogeneous Environment with Label Noise

As seen in previous experiment in Section 6, regular AsynFL presents unsatisfactory training convergence behavior in HeDC. To better thoroughly understand the origin of such phenomenon, we

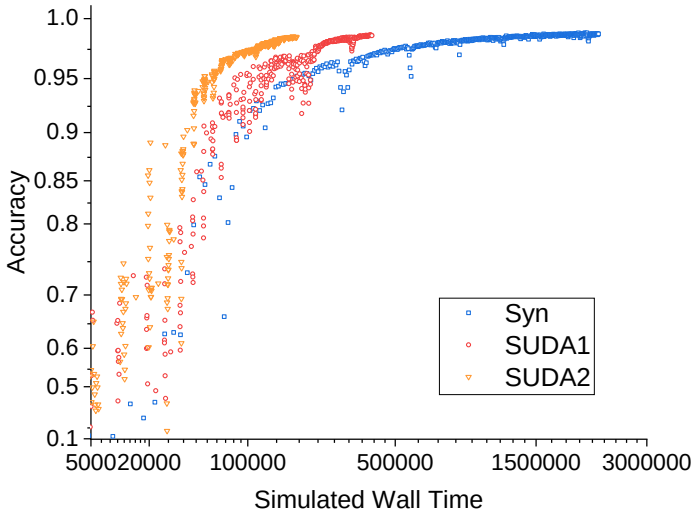


Figure 6.10: Accuracy over wall time.

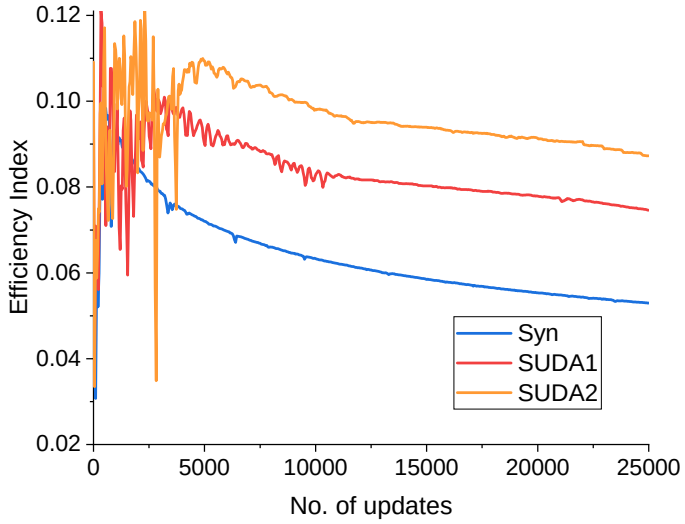


Figure 6.11: Efficiency comparison.

conduct an experiment in **HeDC with label noise**. The experiment setup mostly follows the setting in Section 6. For participant data distribution, we adopt the one as in scenario B. For heterogeneous response distribution, we set 18 participants (ID 82-90 and ID 92-100) to be slower participants in *SP*, and all other be regular participants. Consequently, participant ID "81" and "91" will be the

only regular participants holding training images for MNIST digits “8” and “9”, respectively. To inject label noise, we switch the labels of all training images of these two participants.

The intention is as follows: for digits “8” and “9”, each digit will have 9 participants holding its training data with correct labels, but be slower in temporal response distribution, and one participant with fast response, but has wrong/noisy labels. If the learning method could correctly balance presence of all participants such that the genuine learning objective is approached, the 10% label noise for digits “8” and “9” should not greatly influence the model performance. However, if the learning method could not balance the presence of the faster participants with wrong labels, letting them be excessively represented and causing model convergence drift, then the model performance could be dramatically deteriorated. Since label noises also exist in nowadays’ learning tasks, this experiment as well represents a practical scenario where label noise presents simultaneously with the appearance of heterogeneous environment.

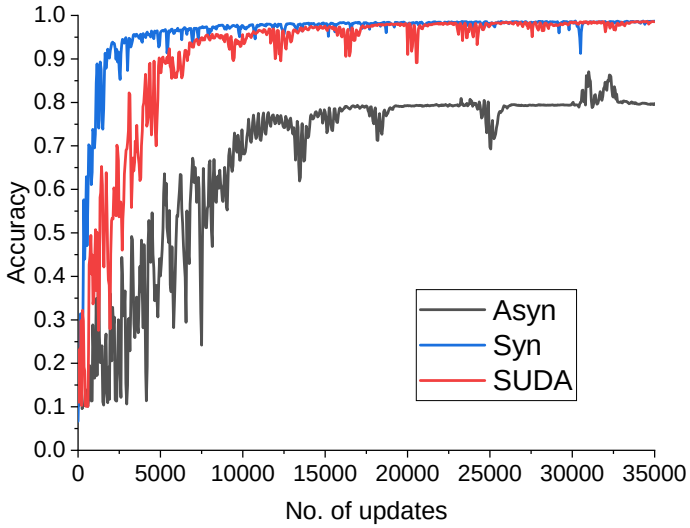


Figure 6.12: Accuracy over number of participant updates.

For this experiment, we increase the total number of updates to 35000 considering the increased training difficulty caused by interference of the injected label noises. As presented in Figure 6.12, Asyn halts at only about 0.8 accuracy and fails again in this HeDC scenario. As expected,

its permissive decoupled learning feature could not regulate the immoderate representation of participants with faster responses but with wrong labels, thus greatly hampers the original learning objective and impairs the model convergence. On the contrary, the proposed Fed-SUDA approach could still converges to the desired learning objective, providing comparable accuracy with Syn, meanwhile keeping a near-asynchronous organization for saving training time. This experiment assists a more comprehensive insight into why AsynFL could be incapable in HeDC and needs to be substituted by approaches like Fed-SUDA and demonstrates the usability of Fed-SUDA in HeDC even with label noises.

Additional Experiment on CIFAR-10

Table 6.3: Training result

Method	Accuracy Milestone	Timestamp
Syn	0.70	4.45×10^6
Fed-SUDA	0.70	0.97×10^6

Our additional experiments on CIFAR-10 dataset intend to show that both the proposed HALE-Fed framework and Fed-SUDA algorithm are general for dataset larger than MNIST. Here, we integrate the participant data distribution in Scenario A with the participant response distribution setting in Scenario B to generate a more general heterogeneous environment. Since the unreliability of AsynFL in HeDC, we focus on the performance of SynFL and Fed-SUDA. In Table 6.3, we record the training timestamp where each method achieves the 0.70 accuracy milestone (the earliest moment when the testing accuracy after each training round has been continually no less than 0.70 in five consecutive training rounds). The proposed Fed-SUDA once again achieves a good **4.6**x speedup.

Note that in our experiments with respect to both MNIST and CIFAR-10, we are not intending to compete with the state-of-the-art accuracy achieved with complex and specialized network design and training tricks. Instead, we intentionally utilize relatively lightweight network structures (a few convolutional and fully-connected layers), which is more likely to appear in realistic FL process where end-device capabilities are constrained. The experiments demonstrate the reliability of the proposed approaches to practical FL process. As a result, Table 6.3 once again demonstrates the efficacy and great training efficiency of the proposed Fed-SUDA method.

Summary

In this chapter, we propose the HALE-Fed framework, which provides SMC-alike technique for participant-level privacy protection to asynchronous FL, and the Fed-SUDA algorithm built upon HALE-Fed, as a reliable asynchronous FL training approach in heterogeneous environments even when label noises exist. Our experiments verify the efficacy of both HALE-Fed and Fed-SUDA and show that they significantly improve training efficiency of federated learning meanwhile avoiding issues in heterogeneous environments.

CHAPTER 7: ASYNCHRONOUS DISTRIBUTED STOCHASTIC GRADIENT DESCENT WITH NON-IID DATA AND HETEROGENEOUS PARTICIPANTS

Distributed learning aims at providing collaborative learning opportunity from decentralized participants. With rapid increase of data generation and often spread data possession nowadays, larger scale distributed data analysis and learning tasks are becoming more and more common. Some significant characteristics of such contemporary learning missions emerge, including non-IID participant data and participant heterogeneity. The former is mostly due to that participants nowadays are often the distinct or even unique data origin; and the later is due to different participant computing and networking capabilities in distributed environment. Similarly as in previous Chapter, we refer it as "**He**terogeneous **D**ata and **C**apability" ("HeDC") when both situations present.

Meanwhile, typical application of distributed learning (DL) could be mainly categorized into two types: (1) federated learning (FL) which keeps private user data local and often assumes a large number of participants with less reliability and non-IID user data [60]; (2) non-FL distributed learning with usually less number of more reliable participants. In fact, both types could experience HeDC. For FL, this is because both non-IID data and participant heterogeneity appear frequently due to its participant scale. For non-FL DL, although traditionally it is often conducted in supercomputer or GPU cluster environment with homogeneous participants possessing IID data, modern distributed learning could easily happen among multiple agencies or data centers as unique data origins, such that their computing and networking capability and data distribution could be largely different. Therefore in this work, we do not explicitly distinguish FL and non-FL distributed learning unless otherwise stated, and refer distributed learning as the general name where HeDC could present.

In perspective of organization form, DL can be majorly categorized into synchronous and asynchronous cases, where the later one intends to improve efficiency by removing training synchronization barriers. With the ever increasing task urgency, learning tasks nowadays including time-critical ones could greatly benefit from asynchronous DL and the necessity of asynchronous organization keeps raising higher. However, asynchronous DL with HeDC needs special consideration. Different with synchronous case where a global participant sampling strategy can be easily controlled, the uncontrolled asynchronous nature could bring bias to participant sampling due to participant heterogeneity, which in combination with non-IID data distribution could cause potential model convergence shift. Therefore, both algorithm design and corresponding theoretical analysis should attend to such inevitable situations. However, to our best knowledge, currently there is a lack of work specifically concentrating on algorithmic design and theoretical analysis for **A**asynchronous algorithms for **N**on-convex problems with **N**on-IID data and **P**articipant heterogeneity (shorted as the "**ANNP**" problem, i.e., non-convex asynchronous distributed learning with HeDC), which comes this work. We do not require problem convexity because non-convexity represents a more general and difficult case for theoretical analysis, and it includes the most general family of neural network structures that are widely used nowadays. Our main contributions in this work are:

- We provide in-depth understanding and causal factor analysis of the model shift phenomenon in asynchronous distributed learning with HeDC condition.
- We propose the HP-ASGD algorithm towards solving model shift issue for general asynchronous distributed learning with HeDC condition and provide according convergence analysis.
- Our convergence result allows choosing important algorithm parameters such as the number of gradients used in a round from a relaxed range instead of a single value during training, making it more practical for actual applications.

In this work, we aim to provide a stochastic gradient descent algorithm and its convergence analysis for a class of distributed learning scenarios with ANNP setting. The ANNP scenario is common nowadays and can happen in both federated learning and other distributed learning process. We now formulate the goal of the learning process into the following minimization problem of $f(x)$:

$$\min_{x\in\mathbb{R}^d} f(x) = \min_{x\in\mathbb{R}^d} \sum_{n=1}^{N} p_n f_n(x), \tag{7.1}$$

where $\{p_n\}_{n=1}^N$ are weights of $\{f_n\}_{n=1}^N$, which are denoted as *p-weight*. There are in total N participants in the learning process. $\mathbb{R}^d$ represents the overall parameter space, and x represents one setting of the network parameters. Each $f_n(x)$ is defined as follows:

$$f_n(x) = \mathbb{E}_{\zeta_n} F(x;\zeta_n)$$

$F(\cdot)$ here is the loss function of the target model. ζ_n refers to the collections of random variables involved in local computing of participant n and $\mathbb{E}$ represents expectation. $\{f_n\}_{n=1}^N$ are smooth but not necessarily convex. The dissimilarity of $f_n(x)$ for each participant n is derived from the non-IID data assumption, which is a key characteristic of this scenario and brings more pressure to the theoretical convergence analysis. In this work, we assume **partial participation**, which means that for the k-th model update round, the server selects M_k number of gradients from participants to execute the model update and not all participants are required to join. This assumption is directly implied by the ANNP setting, since the asynchronous organization generally utilizes the partial participation scheme to alleviate the straggler issue.

p-weight, u-weight and Their Connections with Model Shift.

The process of learning often contains presentations of randomness. With respect to distributed learning, one possible randomness is the widely used stochastic gradient generation basing on sampled data mini-batches. We refer this as **"Data sampling randomness"** (short as D-randomness) which also exists in many other forms of machine learning. The other form of randomness is from the participant sampling occurs in the partial participation scheme of distributed learning, we refer this as the **"Participant sampling randomness"** (short as P-randomness).

In contrary to the well studied D-randomness, P-randomness is still lacking studies to provide more in-depth description of its effects and influences to the model convergence especially in combination with other conditions. On one aspect, this is because P-randomness may not appear in other forms of machine learning, for instance, non-distributed learning does not have P-randomness as there is solely a single participant in the system. On the other aspect, P-randomness in traditional distributed learning also may neither present nor have direct impact on model convergence under certain circumstances. For instance, full participation distributed learning does not have P-randomness since all participants are required to attend each round. And in synchronous distributed learning with IID data and partial participation, P-randomness may present but could not directly influence the convergence of model due to the unified identical data distribution.

However, as mentioned in Section 1, contemporary DL often possesses key features including Non-IID data, partial participation and participant heterogeneity. We argue that the combination of these key features dramatically enlarges the influence of P-randomness to model convergence. Without treating it properly, model shift could occur. In [43], it also raises the concern that the participant sampling policy and corresponding gradient averaging policy are significant to model convergence. However, it avoids the problem by assuming when sampling first k responses from u participants in each round for synchronous DL with partial participation, it follows a uniform

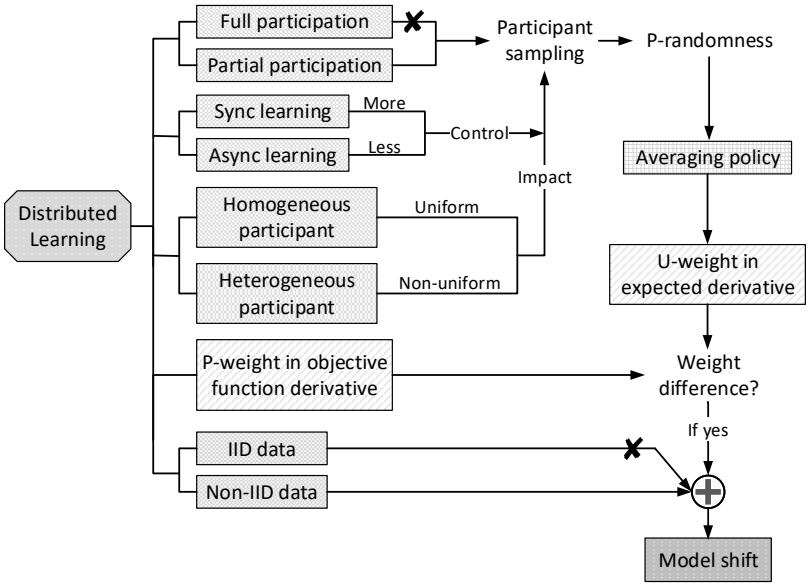


Figure 7.1: Causal factors of model shift with respect to difference of p-weight and u-weight distributions.

Table 7.1: Notations in this work

Notation	**Description**
x^*	The optimal solution
$\mathbb{E}_{\zeta_{D,P}}(\cdot)$	Expectation on Data(D) and Participant(P) related random variables
$\tilde{G}_n(x;\zeta)$	A gradient w.r.t. $f_n(x)$, short as $\tilde{G}_n$
$G_n(x;\zeta)$	An update gradient uploaded to server by participant n, short as G_n
K	Number of training rounds
M_k	Number of gradients used in k-th round
$\tau_{k,m}$	Delay of the m-th gradient in k-th round
γ_k	Learning rate in k-th round

distribution. Such an assumption is less practical when having participant heterogeneity. In this work, we assume a more general sampling distribution, discuss its discrepancy with the p-weight distribution, and provide according algorithmic design and convergence analysis. We also provide a comprehensive factor depict of potential model shift due to different characteristics of DL as in Figure 7.1 and elaborate it more in the following.

P-randomness happens when partial participation is applied. On what extent this sampling process

could be controlled depends on other properties of the learning process. For synchronous cases, we generally have better control to the participant sampling process due to the synchronization barriers. For instance, [42] proposes to control the participant sampling by following the same weight distribution as in the overall objective function. In contrary, for asynchronous cases, participant sampling is mostly implicitly impacted by the underlying deployment environment, *i.e.*, the participant heterogeneity becomes its dominant factor.

When data distribution is IID, P-randomness has no apparently direct impact on model convergence as participants with IID data could mutually compensate each other. However, when data is non-IID, participants are unique and participant sampling discrepancy could cause model convergence shift. Thus in ANNP setting, the role of P-randomness should not be under-estimated. We formally state this problem in the following with notations in Table 6. To achieve the minimum as in Formula 7.1, we need to estimate the gradient:

$$\nabla f(x) = \sum_{n=1}^{N} p_n \nabla f_n(x)$$

The weight distribution here is the same **p-weight** as in Formula 7.1. Now we pay attention to the expectation over all random variables (with respect to both D-randomness and P-randomness, denoted as $\zeta_{D,P}$) of the local derivative as used in SGD algorithm (using an averaging gradient policy without scaling) in ANNP setting: (in this case, $G_m = \tilde{G}_m$)

$$\mathbb{E}_{\zeta_{D,P}} \frac{1}{M_k} \sum_{m=1}^{M_k} G_m = \mathbb{E}_{\zeta_{D,P}} \frac{1}{M_k} \sum_{m=1}^{M_k} \tilde{G}_m = \sum_{n=1}^{N} u_n \nabla f_n(x) \tag{7.2}$$

We refer $\{u_n\}_{n=1}^{N}$ in the expected derivative as the **u-weight**. In general, both the selected averaging policy (including gradient scaling) and P-randomness could jointly affect the u-weight. Specifically in Equation 7.2, as a speical case of u-weight, we properly select the averaging policy for algorithm design purpose and also for the purpose that the influence of the P-randomness to the u-weight can

be separately analyzed here in this specific setting. Suppose the set of participant IDs attending an update round is allowed to be a multiset (*i.e.*, we may have more than one non-duplicate gradients from a participant in one round), by counting participant gradients used in training rounds, we have: $u_n = \mathbb{E}(fre_n) = \mathbb{E}\left(\frac{N_n}{M_A}\right)$ in Equation 7.2 where M_A is the accumulated number of gradients used up to current round, $\{N_n\}_{n=1}^N$ are accumulated number of gradients from each participant, $fre_n = N_n/M_A$ represents frequentness. If we temporarily exclude other potential minor influences from system regulations, and concentrate on the foremost factor affecting u-weight in Eqn. 7.2, we have:

$$u_n = \mathbb{E}\left(\frac{N_n}{M_A}\right) \sim \frac{1/s_n}{\sum_{n=1}^N 1/s_n} = \frac{1}{s_n \sum_{n=1}^N 1/s_n} \quad \forall n \in [1,N]$$

where $s_n = \mathbb{E}_{\zeta_n}(r_n + c_n)$ is the expected response time of each participant, in which r_n and c_n are its computation time and communication time in a round. That is, u-weight in Equation 7.2 is mostly affected by the P-randomness and the foremost influential factor to u-weight in this case is usually determined by the response speed expectations of all participants. In practice, the system could enforce other regulations influencing the P-randomness and consequently the u-weight distribution. Such as, there could be a participating permission policy that partially regulates if a participant is allowed to attend the model update process at a certain moment. These could also impact u-weight and make direct analytical approximation of u-weight unlikely. In this regard, we proposes to record and utilize $fre_n = N_n/M_A$ (the observed appealing frequency of participant n's update up to current) in practical deployment to actualize u_n in Eqn. 7.2. In later portions of this chapter, $\{u_n\}_{n=1}^N$ is referring to this set of weight in the expected derivative in Eqn. 7.2.

Recall that to ensure the correctness of the algorithm, we must require the global gradient in DL to be unbiased. However, due to the difference of p-weight and weight in the expected derivative caused by ANNP, the global gradient used in general asynchronous learning is in fact biased and causes a convergence shift. Thus the model updating discrepancy between the original gradient $\sum_{n=1}^N p_n \nabla f_n(x)$ and the biased target gradient must be compensated in the algorithm design for

a correct convergence result. Based on this practical model shift concern in ANNP setting, we consequently propose HP-ASGD.

HP-ASGD: Algorithm Design.

We present HP-ASGD as an asynchronously paralleled SGD for distributed learning with non-IID data and participant heterogeneity, which is shown in Algorithm 1. Comparing to a typical ASGD, HP-ASGD corrects the global gradient generation by scaling the local gradient function with the factor of p_n/fre_n to statistically eliminate the shift caused by discrepancy between p-weigh and u-weight. With such correction by scaling, it is equivalent to defining an uploaded gradient G_m from participant m as $G_m = \frac{p_m}{fre_m}\tilde{G}_m$. Thus, the expectation of the updating vector over all random variables in HP-ASGD algorithm in ANNP setting for $\forall x$ now becomes:

$$\begin{aligned}
&\mathbb{E}_{\zeta_{D,P}}\frac{1}{M_k}\sum_{m=1}^{M_k} G_m(x;\zeta) = \mathbb{E}_{\zeta_P}\frac{1}{M_k}\sum_{m=1}^{M_k}\mathbb{E}_{\zeta_D}\frac{p_m}{fre_m}\tilde{G}_m(x;\zeta)\\
&= \mathbb{E}_{\zeta_P}\frac{1}{M_k}\sum_{m=1}^{M_k}\nabla\frac{p_m}{fre_m}f_m(x) = \frac{1}{M_k}\sum_{m=1}^{M_k}\mathbb{E}_{\zeta_P}\nabla\frac{p_m}{fre_m}f_m(x)\\
&= \frac{1}{M_k}\sum_{m=1}^{M_k}\sum_{n=1}^{N} u_n\nabla\frac{p_n}{u_n}f_n(x) = \frac{1}{M_k}\sum_{m=1}^{M_k}\sum_{n=1}^{N}\nabla p_n f_n(x)\\
&= \frac{1}{M_k}\sum_{m=1}^{M_k}\nabla f(x) = \nabla f(x)
\end{aligned}$$

In this way, the induced potential model shift due to ANNP setting is corrected internally. Thus, the proposed algorithm HP-ASGD should work as anticipated as a SGD style algorithm for asynchronous learning in a HeDC environment.

Algorithm 7 HP-ASGD

Require: x_1,K,$\{\gamma_k\}_{k=1}^{K}$,$\{M_k\}_{k=1}^{K}$
Output: x_{K+1}
• **Server:**
continually receives gradients $G^{\{k,n\}}$ from participants in the background, stores in server buffer, records its origin (*i.e.*, participant n) and according model version (*i.e.*, k) used for computing.
while $k \leq K$: **do**
  if len(buffer) $\geq M_k$ **then**
    randomly choose $\{G_m\}_{m=1}^{M_k}$ gradients satisfying staleness bound from buffer.
    $x_{k+1} = x_k - \gamma_k \frac{1}{M_k} \sum_{m=1}^{M_k} G_m$
    remove $\{G_m\}_{m=1}^{M_k}$ from buffer, $k = k+1$
  end if
end while
• **Each participant** n**:**
When be able and allowed to participate in learning:
    obtain current model $\{x_k, k\}$ from the server,
    sample a batch of data from local data, compute $G^{\{k,n\}} = \nabla \frac{p_n}{fre_n} F(x;\zeta_n) = \frac{p_n}{fre_n} \tilde{G}_n(x;\zeta)$
    send $G^{\{k,n\}}$ to server

HP-ASGD: Convergence Analysis.

We now provide the theoretical analysis with the following assumptions. These assumptions are practical and analogous versions of these are also used in other works for convergence analysis [46, 43] related to machine learning:

- **Local gradient is unbiased (w.r.t. local objective)**:

$$\nabla f_n(x) = \mathbb{E}_{\zeta_D}[\tilde{G}_n(x;\zeta)] \quad \forall x, \forall n$$

- **Variance of uploaded gradient is bounded**:

$$\mathbb{E}_{\zeta_{D,P}} \left(\|G_n(x;\zeta) - \nabla f(x)\|^2 \right) \leq \sigma_1^2 \quad \forall x, \forall n$$

- **Lipschitzian gradient**:

$$\|\nabla f_n(y) - \nabla f_n(x)\| \leq L\|y - x\| \quad \forall n, \forall x, y$$

- **Random variables are independent**: all random variables involved in the algorithm are independent.
- **Staleness is bounded**: The gradient staleness (or delay), which is the version number difference between model used for computing and current model for any gradient used in arbitrary k-th round is uniformly upper-bounded, i.e. $\tau_{k,m} \leq T \quad \forall k, \forall m$.
- M_k **is lower-bounded**: $M_k \geq M_L \ \forall k$, where $M_L \geq 1$. Notice that this is a **naturally satisfied** assumption as $M_k \geq 1$ is always satisfied. In practice, we can set the lower bound $M_L \geq 1$ based on needs.

With these assumptions, the main convergence results for HP-ASGD under the assumed scenario is an ergodic convergence result in Theorem 7.0.1. It is followed by Corollary 7.0.1.1 which gives a more direct result for the upper bound of the averaged value of $\mathbb{E}(||\nabla f(x_k)||^2)$ over training rounds, as an illustration of how $\|\nabla f(x_k)\|$ statistically vanishes along with the training process. Our convergence analysis in Theorem 7.0.1 and Corollary 7.0.1.1 are inspired by the convergence analysis of algorithm for computer network in [46]. However, there are several differences and improvements in our analysis as listed below:

- Convergence analysis in [46] is for distributed SGD in traditional DL. But ours is for the new HP-ASGD algorithm for both FL and non-FL related distributed learning.
- Convergence analysis in [46] does not consider dissimilarity in local objective functions due to its assumption. On the contrary, we consider such dissimilarities by assuming different $\{f_n(x)\}_{n=1}^N$

due to non-IID participants' data and further consider heterogeneous participants. These change the problem basis and bring challenges to our convergence analysis.

- Proof in [46] requires a fixed value for both the learning rate ($\gamma_k = \gamma, \forall k$) and the numbers of gradients for partial participation ($M_k = M, \forall k$) during training which may not be optimal or practical in certain applications. We relax these important parameters in our proof from a fixed value to a range of values.

We now state Theorem 7.0.1 and its Corollary.

Theorem 7.0.1. *With the listed assumptions and the learning rate sequence* $\{\gamma_k\}_{k=1,\cdots,K}$ *satisfies*

$$\gamma_k L + 2L^2 T \gamma_k \sum_{\kappa=1}^{T} \gamma_{k+\kappa} \leq 1$$

We will get the ergodic convergence rate for Algorithm 1 as

$$\frac{1}{\sum_{k=1}^{K} \gamma_k} \sum_{k=1}^{K} \gamma_k \mathbb{E}\left(\|\nabla f(x_k)\|^2\right) \leq \frac{2(f(x_1) - f(x^*)) + \sum_{k=1}^{K} \left(\frac{\gamma_k^2 L}{M_L} + 2L^2 \gamma_k \sum_{j=k-T}^{k-1} \gamma_j^2\right) \sigma_1^2}{\sum_{k=1}^{K} \gamma_k}$$

Proof. From assumption, all $\{f_n(x)\}_{n=1}^{N}$ have Lipschitzian gradient with L, from the definition of $f(x)$, it is straight forward to deduce that $f(x)$ also has Lipschitzian gradient with L, thus we can have

$$\begin{aligned} f(x_{k+1}) - f(x_k) &\leq \langle \nabla f(x_k), x_{k+1} - x_k \rangle + \frac{L}{2} \|x_{k+1} - x_k\|^2 \\ &= -\left\langle \nabla f(x_k), \gamma_k \sum_{m=1}^{M_k} \frac{1}{M_k} G_m(x_{k-\tau_{k,m}}; \zeta_{k,m}) \right\rangle + \frac{\gamma_k^2 L}{2} \left\| \sum_{m=1}^{M_k} \frac{1}{M_k} G_m(x_{k-\tau_{k,m}}; \zeta_{k,m}) \right\|^2 \end{aligned}$$

Then with expectation taken on $\zeta_{D,P}$,

$$\mathbb{E}_{\zeta_{D,P}}\left(f(x_{k+1})-f(x_k)\right) \leq -\gamma_k \langle \nabla f(x_k), \mathbb{E}_{\zeta_{D,P}}\left(\frac{1}{M_k}\sum_{m=1}^{M_k} G_m(x_{k-\tau_{k,m}};\zeta_{k,m})\right)\rangle + \frac{\gamma_k^2 L}{2}\mathbb{E}_{\zeta_{D,P}}\left(\left\|\sum_{m=1}^{M_k}\frac{1}{M_k}G_m(x_{k-\tau_{k,m}};\zeta_{k,m})\right\|^2\right)$$

$$\mathbb{E}_{\zeta_{D,P}}\left(f(x_{k+1})-f(x_k)\right) \leq -\gamma_k \langle \nabla f(x_k), \frac{1}{M_k}\sum_{m=1}^{M_k} \mathbb{E}_{\zeta_{D,P}}\left(G_m(x_{k-\tau_{k,m}};\zeta_{k,m})\right)\rangle + \frac{\gamma_k^2 L}{2}\mathbb{E}_{\zeta_{D,P}}\left(\left\|\sum_{m=1}^{M_k}\frac{1}{M_k}G_m(x_{k-\tau_{k,m}};\zeta_{k,m})\right\|^2\right)$$

For each G_m, which is the uploaded gradient of one participant, it experiences both D-randomness and P-randomness, thus its expectation on both randomnesses becomes (notice that ζ_D is equivalent as ζ_m, $\forall m$):

$$\begin{aligned}\mathbb{E}_{\zeta_{D,P}}G_m(x) &= \mathbb{E}_{\zeta_P}\mathbb{E}_{\zeta_D}G_m(x) = \mathbb{E}_{\zeta_P}\mathbb{E}_{\zeta_D}\frac{p_m}{fre_m}\tilde{G}_m = \mathbb{E}_{\zeta_P}\frac{p_m}{fre_m}\nabla f_m(x)\\ &= \sum_{n=1}^{N} u_n \nabla \frac{p_n}{u_n} f_n(x) = \sum_{n=1}^{N}\nabla p_n f_n(x) = \nabla f(x)\end{aligned}$$

Similarly, we have $\mathbb{E}_{\zeta_{D,P}}\frac{1}{M_k}\sum_{m=1}^{M_k} G_m(x) = \nabla f(x)$. This is an important deduction in our proof that brings two-fold significance to the following part of the proof:

- This guarantees that the update gradient we used is unbiased and correct for global gradient descent process. This is a fundamental and foremost basis that enables the convergence proof of HP-ASGD for ANNP problem.
- By considering both D-randomness and P-randomness, we connect the local gradients ap-

peared in the partial participation directly to the gradient of the unified global objective function $\nabla f(x)$ instead of the local objective function $\nabla f_n(x)$, this significantly helps reducing the upper-bound regulation form of $\mathbb{E}_{\zeta_{D,P}}(f(x_{k+1})) - f(x_k)$ from an otherwise difficult-to-estimate one to the easier form in the following. This approach enables a convergence result without any residue term that is not controlled by K.

With such consideration, even with a significantly different assumption and application scenario, we are able to alternate the form of the previous inequality to the following one which is similar to a form appearing in the proof of [46]. In brief, the previous inequality becomes,

$$\mathbb{E}_{\zeta_{D,P}}(f(x_{k+1})) - f(x_k) \leq -\gamma_k \langle \nabla f(x_k), \frac{1}{M_k} \sum_{m=1}^{M_k} \nabla f(x_{k-\tau_{k,m}}) \rangle + \frac{\gamma_k^2 L}{2} \mathbb{E}_{\zeta_{D,P}} \left(\left\| \sum_{m=1}^{M_k} \frac{1}{M_k} G_m(x_{k-\tau_{k,m}}; \zeta_{k,m}) \right\|^2 \right)$$

which is,

$$\mathbb{E}_{\zeta_{D,P}}(f(x_{k+1})) - f(x_k) \leq -\gamma_k \langle \nabla f(x_k), \frac{1}{M_k} \sum_{m=1}^{M_k} \nabla f(x_{k-\tau_{k,m}}) \rangle + \frac{\gamma_k^2 L}{2M_k^2} \mathbb{E}_{\zeta_{D,P}} \left(\left\| \sum_{m=1}^{M_k} G_m(x_{k-\tau_{k,m}}; \zeta_{k,m}) \right\|^2 \right)$$

Since it is known that: $\langle x,y\rangle = \frac{1}{2}\left(\|x\|^2+\|y\|^2-\|x-y\|^2\right)$, we have

$$
\begin{aligned}
&\mathbb{E}_{\zeta_{D,P}}(f(x_{k+1}))-f(x_k)\\
&\leq -\frac{\gamma_k}{2}\left(\|\nabla f(x_k)\|^2+\left\|\frac{1}{M_k}\sum_{m=1}^{M_k}\nabla f(x_{k-\tau_{k,m}})\right\|^2-\underbrace{\left(\left\|\nabla f(x_k)-\frac{1}{M_k}\sum_{m=1}^{M_k}\nabla f(x_{k-\tau_{k,m}})\right\|^2\right)}_{T_1}\right)\\
&\quad+\frac{\gamma_k^2 L}{2M_k^2}\underbrace{\mathbb{E}_{\zeta_{D,P}}\left(\left\|\sum_{m=1}^{M_k}G_m(x_{k-\tau_{k,m}};\zeta_{k,m})\right\|^2\right)}_{T_2}
\end{aligned}
$$

the remaining portion of the proof can follow similar approach as in the proof of Theorem 1 in [46] to obtain upper bound estimation for both T_1 and T_2. Interesting readers could read the according section in [46] for references. Eventually, we could achieve the induction which conclude the proof (with abbreviating $\mathbb{E}_{\zeta_{D,P}}$ as $\mathbb{E}$):

$$
\frac{1}{\sum_{k=1}^K\gamma_k}\sum_{k=1}^K\gamma_k\mathbb{E}\left(\|\nabla f(x_k)\|^2\right)\leq\frac{2(f(x_1)-f(x^*))+\sum_{k=1}^K\left(\frac{\gamma_k^2 L}{M_L}+2L^2\gamma_k\sum_{j=k-T}^{k-1}\gamma_j^2\right)\sigma_1^2}{\sum_{k=1}^K\gamma_k}
$$

□

Corollary 7.0.1.1. *With the listed assumptions and set the learning rate γ_k to be within the range*

$$
\frac{1}{C}\sqrt{\frac{f(x_1)-f(x^*)}{LK\sigma_1^2}}\leq\gamma_k\leq\sqrt{\frac{f(x_1)-f(x^*)}{LK\sigma_1^2}}
$$

where $C\geq 1$ is a selected constant for all k. If the following inequality satisfies

$$
K\geq\frac{4L(f(x_1)-f(x^*))(T+1)^2}{\sigma_1^2}
$$

Then the output of Algorithm 1 satisfies the following ergodic convergence rate:

$$\min_{k\in\{1,\cdots,K\}} \mathbb{E}(||\nabla f(x_k)||^2) \leq \frac{1}{K}\sum_{k=1}^{K}\mathbb{E}(||\nabla f(x_k)||^2) \leq \frac{(4M_L+1)C^2}{M_L}\sqrt{\frac{(f(x_1)-f(x^*))\sigma_1^2 L}{K}}$$

Proof. Set

$$\gamma_{lb} := \frac{1}{C}\sqrt{\frac{f(x_1)-f(x^*)}{LK\sigma_1^2}}$$

and

$$\gamma_{ub} := \sqrt{\frac{f(x_1)-f(x^*)}{LK\sigma_1^2}}$$

From the conditions, we have

$$\gamma_{lb} \leq \gamma_k \leq \gamma_{ub} \leq \frac{1}{2L(T+1)}$$

It follows that

$$\gamma_k L + 2L^2 T\gamma_k \sum_{\kappa=1}^{T}\gamma_{k+\kappa} \leq \frac{1}{2(T+1)} + \frac{2T^2}{4(T+1)^2} < \frac{1}{2}+\frac{1}{2} = 1 \quad \forall k$$

Thus condition in Theorem 3.1 is satisfied, by statement in Theorem 3.1, we have

$$\frac{1}{\sum_{k=1}^{K}\gamma_k}\sum_{k=1}^{K}\gamma_k\mathbb{E}\left(\|\nabla f(x_k)\|^2\right) \leq \frac{2(f(x_1)-f(x^*))+\sum_{k=1}^{K}\left(\frac{\gamma_k^2 L}{M_L}+2L^2\gamma_k\sum_{j=k-T}^{k-1}\gamma_j^2\right)\sigma_1^2}{\sum_{k=1}^{K}\gamma_k}$$

Also since

$$\frac{\gamma_{lb}}{K\gamma_{ub}}\sum_{k=1}^{K}\mathbb{E}(||\nabla f(x_k)||^2) = \frac{1}{KC}\sum_{k=1}^{K}\mathbb{E}(||\nabla f(x_k)||^2) \leq \frac{1}{\sum_{k=1}^{K}\gamma_k}\sum_{k=1}^{K}\gamma_k\mathbb{E}(||\nabla f(x_k)||^2)$$

we have

$$\frac{1}{KC}\sum_{k=1}^{K}\mathbb{E}(||\nabla f(x_k)||^2) \leq \frac{2(f(x_1)-f(x^*))+\sum_{k=1}^{K}\left(\frac{\gamma_k^2 L}{M_L}+2L^2\gamma_k\sum_{j=k-T}^{k-1}\gamma_j^2\right)\sigma_1^2}{\sum_{k=1}^{K}\gamma_k}$$

$$
\begin{aligned}
&\leq \frac{2(f(x_1)-f(x^*))+K\left(\frac{\gamma_{ub}^2 L}{M_L}+2L^2T\gamma_{ub}^3\right)\sigma_1^2}{K\gamma_{lb}} \\
&= \frac{2(f(x_1)-f(x^*))}{K\gamma_{lb}} + \frac{LC\sigma_1^2\gamma_{ub}}{M_L} + 2L^2CT\sigma_1^2\gamma_{ub}^2 \\
&\leq 2C\sqrt{\frac{(f(x_1)-f(x^*))\sigma_1^2 L}{K}} + \frac{C}{M_L}\sqrt{\frac{(f(x_1)-f(x^*))\sigma_1^2 L}{K}} \\
&\quad +2C\sqrt{\frac{(f(x_1)-f(x^*))\sigma_1^2 L}{K}} \\
&= \frac{(4M_L+1)C}{M_L}\sqrt{\frac{(f(x_1)-f(x^*))\sigma_1^2 L}{K}}
\end{aligned}
$$

Thus

$$
\min_{k\in\{1,\cdots,K\}} \mathbb{E}(||\nabla f(x_k)||^2) \leq \frac{1}{K}\sum_{k=1}^{K}\mathbb{E}(||\nabla f(x_k)||^2) \leq \frac{(4M_L+1)C^2}{M_L}\sqrt{\frac{(f(x_1)-f(x^*))\sigma_1^2 L}{K}}
$$

□

The convergence analysis provided by Theorem 7.0.1 and Corollary 7.0.1.1 evidently show that the proposed HP-ASGD algorithm can converge under the considered assumption with a speed of $O(1/K^{\frac{1}{2}})$.

Experiments

We test the HP-ASGD algorithm in a variety of scenarios for verifying its efficacy. Since HP-ASGD is in purpose of rectifying asynchronous SGD algorithm in HeDC environment, we include native asynchronous SGD algorithm (for verifying the correction effect) and synchronous SGD algorithm (for verifying efficiency) as comparative algorithms.

Two representative datasets are used including MNIST and CIFAR-10. We also vary other significant

experiment settings to enhance the coverage of the conducted experiments, including: (1) different total number of participants in experiments (10 and 100 participants); (2) different data distribution and participant heterogeneity; (3) whether or not there exist training data label noises; (4) different M_L setting which regulates at least how many participant generated updates are used for each round of model training.

Experiment-I

The first and foremost result in our experiments is to demonstrate the model shift phenomenon caused by native asynchronous algorithm in HeDC environments. As discussed previously, this phenomenon inevitably causes an eventual erroneous convergence result and compromises the original learning objective. However, the difficult part is that such shift in the independent variable space (the network parameter space) may not be strictly proportionally reflected in the performance metric (accuracy), because how $f(x)$ changes with respect to a specific model shift Δx depends on many problem nature related factors. For native asynchronous algorithm in HeDC, when there is a significant drop in final model performance, it is apparent that a model shift occurred. But even when accuracy deterioration is small, the result from native asynchronous method remains dubious and unacceptable in HeDC environments due to the highly possible existence of model shift. And model performance with respect to such a result could easily exacerbate with even slight changes in problem conditions.

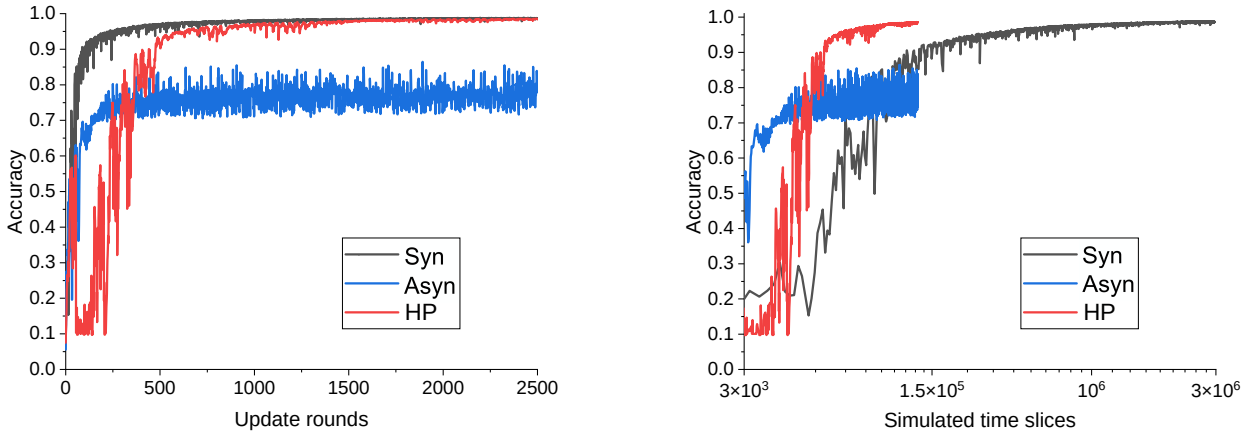
To better illustrate the harmful model shift caused by native asynchronous algorithm, we present the following experiment scenario which (1) commonly exists in practical distributed learning scenarios and (2) reflects obvious model accuracy drop with respect to certain model shift. In general DL tasks, we could frequently encounter problems where the overall training data set contains certain amount of label "noises". This is not only because label noises could commonly

exist in training set especially when dataset is large, but also because data label conflicting naturally and reasonably exists in many practical learning problems. For instance, in a sentence "next-word" prediction problem, non-IID training data from multiple participants will more than likely contain different guidance for similar or even identical input and the intended model shall learn based on the dominating tendency. Similarly, a sentiment predictor shall learn from possibly differentiating or conflicting sentiment labels and try to catch the principal opinion.

These kinds of label noises could slow down the learning process to some extent, but shall not vitally influence the eventual model performance due to their expected much lower proportion than regular data. However, in HeDC environment, if model shift happens and reflects an erroneous convergence target such that the effect of label noises are exaggerated, then not only the convergence objective but also the eventual model performance could be vitally undermined.

We now realize such scenario by intentionally injecting label noises to MNIST dataset. For MNIST, we utilize its 50000 training data for model learning. In this experiment, there are totally 100 participants, with each by sequence take 500 MNIST training data (also by sequence) without overlapping. Thus participants in general have non-IID data. And for each MNIST digit, there are exactly ten participants possessing its data. We inject label noises by perturbing all the data labels of the first participant who holds data with respect to digit 7-9. Thus for these three digits only, there exist 10% label noises. For participants holding data for digit 7-9 but with no label noises, we let their expected response speeds (expected number of responses over a unit time) be generally slower, whereas all other participants in the learning has faster response speeds. Among all participants, the slowest expected response speed is about 20x slower than the fastest one. Since participants have different response speeds and non-IID data, this forms a HeDC environment.

We now examine the performance of our proposed approach HP-ASGD (HP) with respect to native synchronous (Syn) and asynchronous (Asyn) SGD approaches. In this experiment, we try to use



(a) Accuracy over rounds

(b) Accuracy over time slices

Figure 7.2: Accuracy comparison of different approaches.

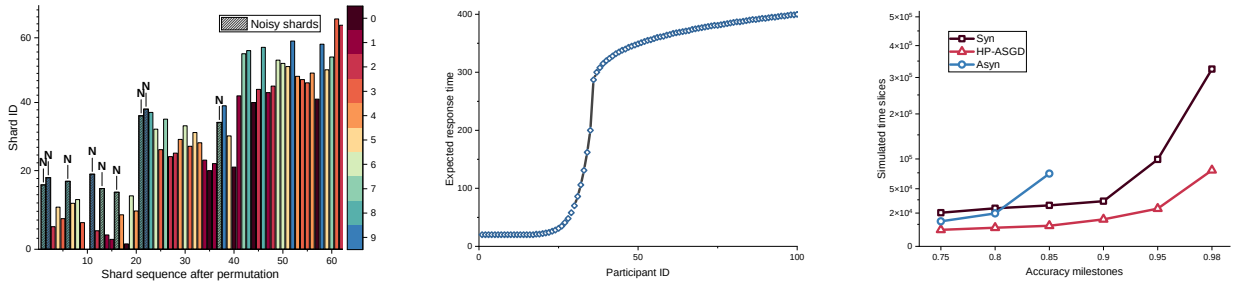
same parameters for all approaches when applicable, including training rounds and learning rate. For "Syn", we set it to utilize $M = 50$ participant generated updates for each round and similar set $M_L = 50$ for "HP" and "Asyn" methods ($M_k \geq M_L$ is still variable with respect to system status at k-th round, but generally be very close to $M_L = 50$). The result of all methods are shown in Figure 7.2 and the conclusions are as follows: (1) As illustrated, "Asyn" method indeed suffers dramatic performance deterioration and clearly indicates the significant model shift. This is because native asynchronous method could not counter the improper over-expression of participants with label noises and fast response speeds. (2) On the contrary, both our proposed method "HP" and native synchronous method "Syn" could still converge to the correct objective in this scenario with similarly good accuracy (about 0.985). (3) However, when considering training efficiency, "Syn" shows apparent and tremendous disadvantage comparing to other two approaches due to the much longer training time. (4) "HP" is the only approach in this scenario that could achieve the desired learning objective while maintaining high training efficiency (speedup over "Syn" is about **20**), and therefore demonstrates its great advantages.

Experiment-II

In this experiment, we test a more general HeDC setting with more intense label noises. For 50000 MNIST training data, we equally divide them into 200 shards, with each shard contains 250 data. The shard generation process is organized that the first 250 data of MNIST digit 0-9 respectively by sequence becomes the first ten shards, and the second 250 data of MNIST digit 0-9 respectively by sequence becomes the second ten shards, so on and so forth. When all 200 shards are generated, they are divided by sequence into 10 groups, with each group containing 20 shards. We then permute the sequence of shards randomly within each group. In such a way, the shard sequence generally becomes random while maintaining a coarse overall tendency that shards with smaller ID tends to remain in front. This is in purpose of generating a more heterogeneous non-IID participant data distribution while keeping the convenience for injecting label noises. In this experiment, we inject label noises by perturbing all labels of the first 750 data from each MNIST digit from 7-9. Thus the noise label percentage now is 15% for each digit in 7-9.

We present the first 60 shard sequence after permutation as an illustrative example of the data organization process in Figure 7.3(a), where a column represents a shard, with height for the original shard ID and the color for corresponding MNIST digit. All shards with label noises are marked with "N" on top. Each participant by sequence take two shards (without replacement) to form their local data which are non-IID. Furthermore, the expected response time for one response of all participants are shown in Figure 7.3(b).

With such a HeDC environment, we once again evaluate all three methods: "Syn", "HP" and "Asyn". In this experiment, we set $M = M_L = 20$ but allow different setting of other parameters that could better suit each method. Notice that the setting of this experiment makes update staleness in learning be generally larger than Experiment I, combining with the more percentage of label noises, this experiment becomes a more difficult situation for all methods. We train all methods for sufficient



(a) Data distribution (b) Expected response time (c) Accuracy milestones

Figure 7.3: Illustrations of experiment setting and results for Experiment-II.

rounds and record the time stamps where a method reaches certain accuracy milestones in the range of 0.75-0.98 (earliest moment when the model achieves certain testing accuracy (with respect to the MNIST testing dataset) after each training round, for five times in a row). We present the result in Figure 7.3(c).

As expected, the result is consistent with Experiment I that "Asyn" fails to converge due to suffering from model shift caused by improper handling of HeDC environment. It could not reach any milestones for accuracy higher than 0.85. Our proposed "HP" still shows much better training efficiency than "Syn" when both methods achieve all accuracy milestones successfully. Combining results in both Experiment I & II, it is apparent that "Asyn" is unreliable in HeDC due to its inevitable model shift and our proposed HP-ASGD could correct such shift while maintain similar high training efficiency with native asynchronous method. We consequently exclude "Asyn" method from further experiments.

Experiment-III

Previous experiments have demonstrated HP-ASGD as a strong substitute for accomplishing both efficiency and correctness in HeDC setting (i.e. ANNP problem). To better understand the property of HP-ASGD, especially its reflection to some parameter selections, we train different variants of

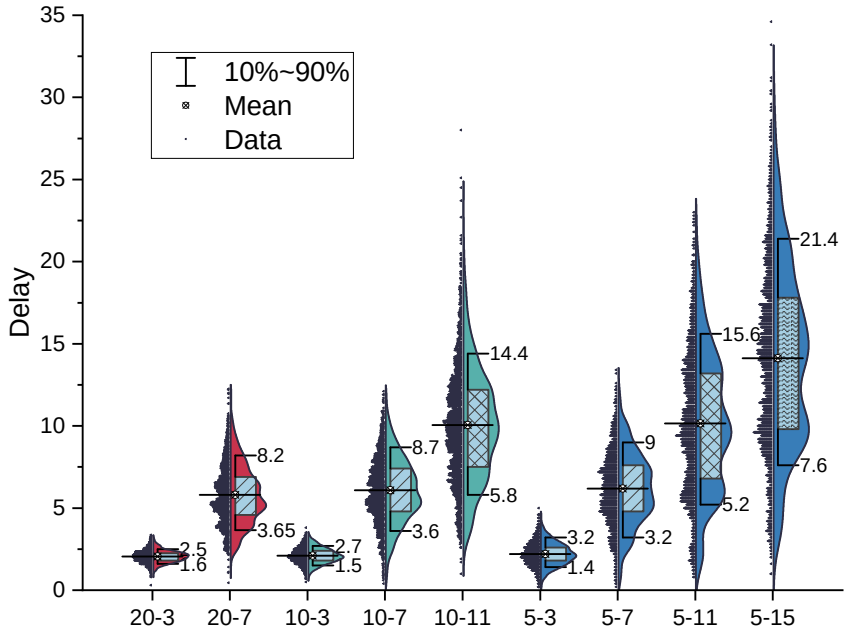


Figure 7.4: Gradient staleness (delay) of different scenarios.

HP-ASGD by varying certain parameters and record our observations. Specifically, we utilize the same experiment setting as in Experiment II except this time without noisy labeling. This enables faster training and could also examine the performance of HP-ASGD under HeDC without label noise along the way.

The key parameters we examine in this experiment include the number of updates used for each training round and the according suitable learning rates. Furthermore, in previous experiments, we generally allow all participants to attend the learning process at any moment without restrictions. In practical deployment, there might be situations where the system would want to restrict an upper-bound threshold of server stored updates or simultaneous participating attendants, possibly due to system storage or processing considerations. We consequently test out this scenario by setting a new parameter "P", which is an upper bound requirement for currently system stored updates plus currently attending participants. This could regulate that neither number of stored updates nor number of simultaneous attending participants at any moment during training exceeding an

upper-bound. When "P" is sufficiently large, this restriction is equivalently degenerated back to our previous experiments. To better observe influence of parameters without other interference, we let number of updates used for each round to be exactly equal to M_L, thus in this experiment $M_k = M_L$ for $\forall k$. We also denote a multiplier C such that $P = C * M_L$. We test different combinations of M_L and C (thus equivalently P), and present the results in Figure 7.4, Tables 7.2 and 7.3. For any unavailable cells (denoted with "-") in Tables 7.2 and 7.3, it means the value of C is too large for the corresponding value of M_L that its performance in experiment shall be similar with the previous choice of C, thus be excluded from testing.

Figure 7.4 presents the distribution of averaged update staleness for all training rounds of each model variant obtained via HP-ASGD with different parameter settings. The horizontal label marks each variant by "M_L-C". For instance, the first horizontal label 20-3 means this variant has $M_L = 20$ and $C = 3$, thus $P = 20 \times 3 = 60$ in the according variant. From it, we observe the following phenomena: (1) For a fixed M_L, larger C thus larger P induces larger averaged update staleness. (2) Among different values of M_L, for a fixed C, the mean of the averaged update staleness is similar, yet the variance of update staleness becomes larger with the decrease of M_L. We then focus on the

Table 7.2: Suitable learning rate for different settings

	$M_L = 5$	$M_L = 10$	$M_L = 20$
$C = 3$	0.1	0.2	0.3
$C = 7$	0.1	0.1	0.05
$C = 11$	0.05	0.05	-
$C = 15$	0.05	-	-

change of suitable learning rates for these different settings. For each setting, we train it with all learning rates in the range $[0.05, 0.3]$ with 0.05 step-size and select the one with the best training efficiency (fastest to achieve 0.98 accuracy milestone). The result is shown in Table 7.2 and the observations are: (1) for each fixed M_L, when C increases, the suitable learning rate becomes smaller. This is expected since by our convergence result the increase of updates staleness could cause a

smaller upper bound for suitable learning rates. (2) For a fixed C value which is small, the increase of M_L could imply a larger suitable learning rate, this tendency becomes less apparent when the fixed C value becomes larger. We speculate that this is caused by the reduce of staleness variance with increase in M_L for a fixed C. For smaller C, the shrinking of the staleness variance is more significant proportionally, thus inducing more influence to suitable learning rate. For lager C, the shrinking of the staleness variance is less significant.

Table 7.3: 0.98 accuracy time stamp for different settings

	$M_L = 5$	$M_L = 10$	$M_L = 20$
$C = 3$	$5.30 * 10^3$	$3.45 * 10^3$	$2.66 * 10^3$
$C = 7$	$2.66 * 10^3$	$2.03 * 10^3$	$2.20 * 10^3$
$C = 11$	$2.63 * 10^3$	$2.44 * 10^3$	-
$C = 15$	$2.60 * 10^3$	-	-

Furthermore, we verify the time stamp for each setting to reach the 0.98 accuracy milestone with its most suitable learning rate as discovered previously, as an indication of its training efficiency. The result of which is presented as in Table 7.3. From it, we may roughly summarize some empirical observations: (1) generally, using a reasonably large M_L may benefit the learning process, as a server update obtained via more participant updates is more accurate in its representativeness. (2) A suitable combination of parameter M_L and C may not certainly appear at their boundary values, a joint consideration of them with the suitable learning rate could be applied when needed by the application scenario.

Experiment-IV

We have seen in previous experiments the performance of HP-ASGD with MNIST dataset for 100 participants (a relatively large scale of participants). In this experiment, we further examine HP-ASGD with smaller scale of participants, which mimics the application scenario for collaborative

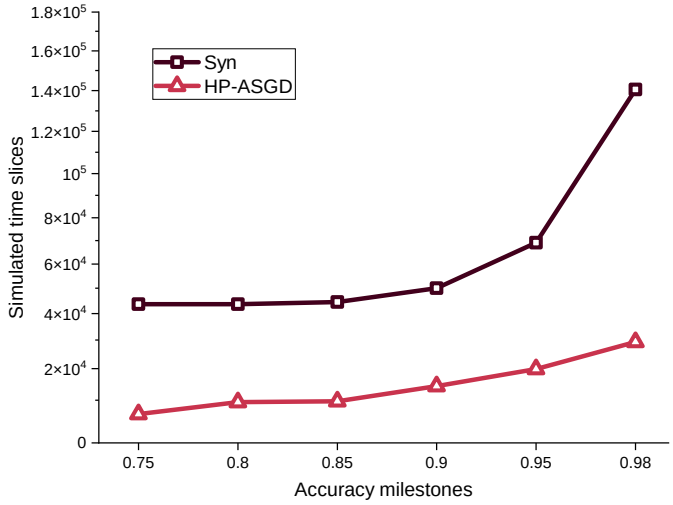


Figure 7.5: Accuracy and efficiency comparison of Experiment-IV.

distributed learning among institutions/data centers (thus with smaller scale of participants).

In this experiment, there are in total 10 participants. We equally divide the MNIST training dataset by sequence into twenty subgroups, each with 2500 images, and each participants randomly pick two subgroups as their local data with no replacement. Thus the participants' data are non-IID. The response speed distribution of participants is analogous to that of the Experiment-II except with 10 total participants this time. Therefore, the overall environment is HeDC. We run "Syn" and "HP" with their suitable learning rate and record the performance. Again, we show the moments when each approach reach certain testing accuracy milestones as in Figure 7.5. For both approaches, they successfully achieves all accuracy milestones, yet the proposed HP-ASGD still maintains a superior training efficiency over Syn. The speedup of HP-ASGD at 0.98 accuracy is **4.8**.

Experiment-V

We further test the performance of HP-ASGD on larger dataet. Specifically, we adopt CIFAR-10 dataset in this experiment. CIFAR-10 also contains 50000 training data, but each training image

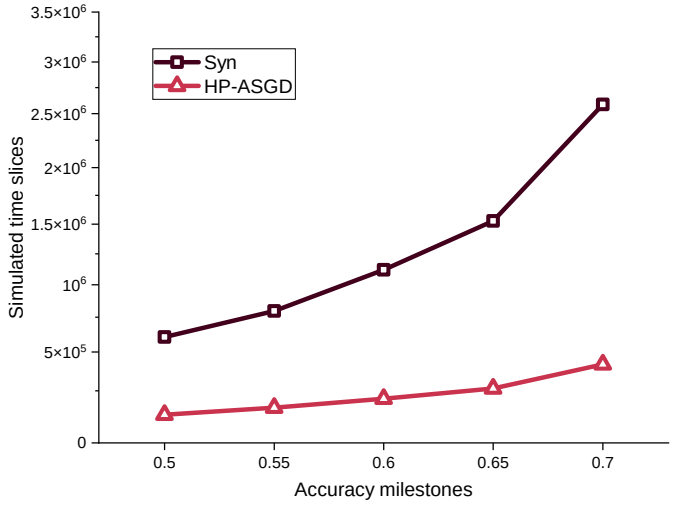


Figure 7.6: Accuracy and efficiency comparison of Experiment-V.

has larger resolution comparing to MNIST, and in overall represents a more challenging learning scenario comparing to MNIST. We set other experiment settings similar as in Experiment-IV to form a HeDC environment and run the two approaches "Syn" and "HP". The accuracy milestones in this experiment are form 0.5 to 0.7 with a 0.05 incremental step due to the increased difficulty of CIFAR-10 dataset. The result in Figure 7.6 is as expected to be consistent with previous experiments. Both approaches converges yet the proposed HP-ASGD still maintains a superior training efficiency over Syn and the speedup of HP-ASGD at 0.7 accuracy is **6.2**.

In all experiments shown, we do not discard participant gradients encountered due to large staleness (*i.e.*, it is equivalent as we set a sufficiently large staleness upper-bound T). This also shows that HP-ASGD algorithm has presented good tolerance to gradient staleness in asynchronous learning.

Summary

In this chapter, we provide in-depth understanding for occurring conditions of the model shift issue in asynchronous learning with non-IID data and participant heterogeneity. we then provide

the designed algorithm HP-ASGD as well as its convergence analysis towards solving such an issue. Extensive experiment results show the efficacy of the proposed algorithm in providing both correctness and high efficiency to ANNP problem comparing to baselines in various learning environment settings.

CHAPTER 8: CONCLUSION

In this dissertation, we establish several approaches including framework, resource management and algorithm design for tackling several challenges faced by the computing facility from the modern data analysis and learning tasks. The hierarchical framework we propose enables collaborative large-scale data analytics among multiple clusters with Apache Spark computing platform. The reinforcement learning based resource management approach we propose allows scheduling data analytical tasks including both general and time-critical jobs to multiple computing clusters for better efficiency and less missing temporal deadline events. Another deep reinforcement learning based resource management approach we design could even be elasticity-compatible and be suitable in an underlying environment of heterogeneous computing clusters. Additionally, we establish a new hybrid approach HALE-Fed and an algorithm Fed-SUDA for more efficient privacy-preserving federated learning and an applicable asynchronous federated learning in a heterogeneous environment. We also propose HP-ASGD and establish its convergence result for non-convex problems, as an asynchronous stochastic gradient descent algorithm for general distributed learning with non-IID data and heterogeneous participants.

Experiment results demonstrate the efficacy of the proposed approaches and show benefits to the efficiency and other significant desires of modern data analytical tasks. We hope to further extend these techniques or develop other skills for enhancing the benefits of such approaches and will continue exploring more possibilities towards further increasing efficiency of modern distributed data analysis and learning tasks.

LIST OF REFERENCES

[1] Omid Alipourfard, Hongqiang Harry Liu, Jianshu Chen, Shivaram Venkataraman, Minlan Yu, and Ming Zhang. Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics. In *NSDI*, volume 2, pages 4–2, 2017.

[2] Enda Barrett, Enda Howley, and Jim Duggan. Applying reinforcement learning towards automating resource allocation and application scalability in the cloud. *Concurrency and Computation: Practice and Experience*, 25(12):1656–1674, 2013.

[3] Andrew G Barto. Temporal difference learning. *Scholarpedia*, 2(11):1604, 2007.

[4] Tekin Bicer, David Chiu, and Gagan Agrawal. A framework for data-intensive computing with cloud bursting. In *2011 IEEE international conference on cluster computing*, pages 169–177. IEEE, 2011.

[5] Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A framework for fast privacy-preserving computations. In *European Symposium on Research in Computer Security*, pages 192–206. Springer, 2008.

[6] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečnỳ, Stefano Mazzocchi, H Brendan McMahan, et al. Towards federated learning at scale: System design. *arXiv preprint arXiv:1902.01046*, 2019.

[7] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for federated learning on user-held data. *arXiv preprint arXiv:1611.04482*, 2016.

[8] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1175–1191, 2017.

[9] Michael Cardosa, Chenyu Wang, Anshuman Nangia, Abhishek Chandra, and Jon Weissman. Exploring mapreduce efficiency with highly-distributed data. In *Proceedings of the second international workshop on MapReduce and its applications*, pages 27–34, 2011.

[10] Zheng Chai, Yujing Chen, Liang Zhao, Yue Cheng, and Huzefa Rangwala. Fedat: A communication-efficient federated learning method with asynchronous tiers under non-iid data. *arXiv preprint arXiv:2010.05958*, 2020.

[11] Guillaume Maurice Jean-Bernard Chaslot Chaslot. *Monte-carlo tree search*. PhD thesis, Maastricht University, 2010.

[12] Lei Chen, Wei Lu, Xiaoping Che, Weiwei Xing, Liqiang Wang, and Yong Yang. Mrsim: Mitigating reducer skew in mapreduce. In *2017 31st International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, pages 379–384. IEEE, 2017.

[13] Ming Chen, Bingcheng Mao, and Tianyi Ma. Efficient and robust asynchronous federated learning with stragglers. In *Submitted to International Conference on Learning Representations*, 2019.

[14] Mingxi Cheng, Ji Li, and Shahin Nazarian. Drl-cloud: Deep reinforcement learning-based resource provisioning and task scheduling for cloud service providers. In *Proceedings of the 23rd Asia and South Pacific Design Automation Conference*, pages 129–134. IEEE Press, 2018.

[15] Christopher M De Sa, Ce Zhang, Kunle Olukotun, and Christopher Ré. Taming the wild: A unified analysis of hogwild-style algorithms. In *Advances in neural information processing systems*, pages 2674–2682, 2015.

[16] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[17] Deepmind. Deepmind. https://deepmind.com/.

[18] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[19] John C Duchi, Sorathan Chaturapruek, and Christopher Ré. Asynchronous stochastic convex optimization. *arXiv preprint arXiv:1508.00882*, 2015.

[20] Xavier Dutreilh, Sergey Kirgizov, Olga Melekhova, Jacques Malenfant, Nicolas Rivierre, and Isis Truck. Using reinforcement learning for autonomic resource allocation in clouds: towards a fully automated workflow. In *ICAS 2011, The Seventh International Conference on Autonomic and Autonomous Systems*, pages 67–74, 2011.

[21] Ahmed Eldawy and Mohamed F Mokbel. Spatialhadoop: A mapreduce framework for spatial data. In *2015 IEEE 31st international conference on Data Engineering*, pages 1352–1363. IEEE, 2015.

[22] Marwa Elteir, Heshan Lin, and Wu-chun Feng. Enhancing mapreduce via asynchronous data processing. In *2010 IEEE 16th International Conference on Parallel and Distributed Systems*, pages 397–405. IEEE, 2010.

[23] Zoltán Gábor, Zsolt Kalmár, and Csaba Szepesvári. Multi-criteria reinforcement learning. In *ICML*, volume 98, pages 197–205, 1998.

[24] R. C. Geyer, T. Klein, and M. Nabi. Differentially Private Federated Learning: A Client Level Perspective. *ArXiv e-prints*, December 2017.

[25] Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10):2222–2232, 2016.

[26] Otkrist Gupta and Ramesh Raskar. Distributed learning of deep neural network over multiple agents. *Journal of Network and Computer Applications*, 116:1–8, 2018.

[27] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604*, 2018.

[28] He Huang, Liqiang Wang, En-Jui Lee, and Po Chen. An mpi-cuda implementation and optimization for parallel sparse equations and least squares (lsqr). *Procedia Computer Science*, 9:76–85, 2012.

[29] Nathaniel Husted, Steven Myers, Abhi Shelat, and Paul Grubbs. Gpu and cpu parallelization of honest-but-curious secure two-party computation. In *Proceedings of the 29th Annual Computer Security Applications Conference*, pages 169–178, 2013.

[30] Chao Jin, Christian Vecchiola, and Rajkumar Buyya. Mrpga: an extension of mapreduce for parallelizing genetic algorithms. In *2008 IEEE Fourth International Conference on eScience*, pages 214–221. IEEE, 2008.

[31] Gregory Kahn, Adam Villaflor, Bosen Ding, Pieter Abbeel, and Sergey Levine. Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE, 2018.

[32] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977*, 2019.

[33] Renuga Kanagavelu, Zengxiang Li, Juniarto Samsudin, Yechao Yang, Feng Yang, Rick Siow Mong Goh, Mervyn Cheah, Praewpiraya Wiwatphonthana, Khajonpong Akkarajitsakul, and Shangguang Wang. Two-phase multi-party computation enabled privacy-preserving federated learning. In *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, pages 410–419. IEEE, 2020.

[34] A Kala Karun and K Chitharanjan. A review on hadoop—hdfs infrastructure extensions. In *2013 IEEE conference on information & communication technologies*, pages 132–137. IEEE, 2013.

[35] Ahmed Khaled, Konstantin Mishchenko, and Peter Richtárik. First analysis of local gd on heterogeneous data. *arXiv preprint arXiv:1909.04715*, 2019.

[36] Hatim Khouzaimi, Romain Laroche, and Fabrice Lefèvre. Reinforcement learning for turn-taking management in incremental spoken dialogue systems. In *IJCAI*, pages 2831–2837, 2016.

[37] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[38] Jakub Konečnỳ, Brendan McMahan, and Daniel Ramage. Federated optimization: Distributed optimization beyond the datacenter. *arXiv preprint arXiv:1511.03575*, 2015.

[39] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[40] Kyong-Ha Lee, Yoon-Joon Lee, Hyunsik Choi, Yon Dohn Chung, and Bongki Moon. Parallel data processing with mapreduce: a survey. *AcM sIGMoD record*, 40(4):11–20, 2012.

[41] Teng Li, Zhiyuan Xu, Jian Tang, and Yanzhi Wang. Model-free control for distributed stream data processing using deep reinforcement learning. *Proceedings of the VLDB Endowment*, 11(6):705–718, 2018.

[42] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. *arXiv preprint arXiv:1812.06127*, 2018.

[43] Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. On the convergence of fedavg on non-iid data. *arXiv preprint arXiv:1907.02189*, 2019.

[44] Yanan Li, Shusen Yang, Xuebin Ren, and Cong Zhao. Asynchronous federated learning with differential privacy for edge intelligence. *arXiv preprint arXiv:1912.07902*, 2019.

[45] Dragos Lia and Mihai Togan. Privacy-preserving machine learning using federated learning and secure aggregation. In *2020 12th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*, pages 1–6. IEEE, 2020.

[46] Xiangru Lian, Yijun Huang, Yuncheng Li, and Ji Liu. Asynchronous parallel stochastic gradient for nonconvex optimization. In *Advances in Neural Information Processing Systems*, pages 2737–2745, 2015.

[47] Guangdeng Liao, Kushal Datta, and Theodore L Willke. Gunther: Search-based auto-tuning of mapreduce. In *European Conference on Parallel Processing*, pages 406–419. Springer, 2013.

[48] Ning Liu, Zhe Li, Jielong Xu, Zhiyuan Xu, Sheng Lin, Qinru Qiu, Jian Tang, and Yanzhi Wang. A hierarchical framework of cloud resource allocation and power management using

deep reinforcement learning. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 372–382. IEEE, 2017.

[49] Teng Liu, Xiaosong Hu, Shengbo Eben Li, and Dongpu Cao. Reinforcement learning optimized look-ahead energy management of a parallel hybrid electric vehicle. *IEEE/ASME Transactions on Mechatronics*, 22(4):1497–1507, 2017.

[50] Zixia Liu, Liqiang Wang, and Gang Quan. Deep reinforcement learning based elasticity-compatible heterogeneous resource management for time-critical computing. In *49th International Conference on Parallel Processing-ICPP*, pages 1–11, 2020.

[51] Zixia Liu, Hong Zhang, Bingbing Rao, and Liqiang Wang. A reinforcement learning based resource management approach for time-critical workloads in distributed computing environment. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 252–261. IEEE, 2018.

[52] Zixia Liu, Hong Zhang, and Liqiang Wang. Hierarchical spark: A multi-cluster big data computing framework. In *Cloud Computing (CLOUD), 2017 IEEE 10th International Conference on*, pages 90–97. IEEE, 2017.

[53] Xiaofeng Lu, Yuying Liao, Pietro Lio, and Pan Hui. Privacy-preserving asynchronous federated learning mechanism for edge network computing. *IEEE Access*, 8:48970–48981, 2020.

[54] Yuan Luo, Zhenhua Guo, Yiming Sun, Beth Plale, Judy Qiu, and Wilfred W Li. A hierarchical framework for cross-domain mapreduce execution. In *Proceedings of the second international workshop on Emerging computational methods for the life sciences*, pages 15–22, 2011.

[55] Yuan Luo, Beth Plale, Zhenhua Guo, Wilfred W Li, Judy Qiu, and Yiming Sun. Hierarchical mapreduce: towards simplified cross-domain data processing. *Concurrency and Computation: Practice and Experience*, 26(4):878–893, 2014.

[56] Hongyi Ma, Liqiang Wang, and Krishanthan Krishnamoorthy. Detecting thread-safety violations in hybrid openmp/mpi programs. In *2015 IEEE International Conference on Cluster Computing*, pages 460–463. IEEE, 2015.

[57] Hongyi Ma, Liqiang Wang, Byung Chul Tak, Long Wang, and Chunqiang Tang. Auto-tuning performance of mpi parallel programs using resource management in container-based virtual cloud. In *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*, pages 545–552. IEEE, 2016.

[58] Shie Mannor and Nahum Shimkin. A geometric approach to multi-criterion reinforcement learning. *Journal of machine learning research*, 5(Apr):325–360, 2004.

[59] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, pages 50–56. ACM, 2016.

[60] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, pages 1273–1282. PMLR, 2017.

[61] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. Inference attacks against collaborative learning. *arXiv preprint arXiv:1805.04049*, 13, 2018.

[62] N Metropolis. Monte carlo method. *From Cardinals to Chaos: Reflection on the Life and Legacy of Stanislaw Ulam*, page 125, 1989.

[63] Fan Mo, Hamed Haddadi, Kleomenis Katevas, Eduard Marin, Diego Perino, and Nicolas Kourtellis. Ppfl: privacy-preserving federated learning with trusted execution environments. In *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*, pages 94–108, 2021.

[64] Jyoti Nandimath, Ekata Banerjee, Ankur Patil, Pratima Kakade, Saumitra Vaidya, and Divyansh Chaturvedi. Big data analysis using apache hadoop. In *2013 IEEE 14th International Conference on Information Reuse & Integration (IRI)*, pages 700–703. IEEE, 2013.

[65] John Nguyen, Kshitiz Malik, Hongyuan Zhan, Ashkan Yousefpour, Michael Rabbat, Mani Malek, and Dzmitry Huba. Federated learning with buffered asynchronous aggregation. *arXiv preprint arXiv:2106.06639*, 2021.

[66] AJ Paverd, Andrew Martin, and Ian Brown. Modelling and automatically analysing privacy properties for honest-but-curious adversaries. *Tech. Rep*, 2014.

[67] Junjie Peng, Xuejun Zhang, Zhou Lei, Bofeng Zhang, Wu Zhang, and Qing Li. Comparison of several cloud computing platforms. In *2009 Second international symposium on information science and engineering*, pages 23–27. IEEE, 2009.

[68] Jia Qian and Lars Kai Hansen. What can we learn from gradients? *arXiv preprint arXiv:2010.15718*, 2020.

[69] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. *Advances in neural information processing systems*, 24:693–701, 2011.

[70] Elsa Rizk, Stefan Vlaski, and Ali H Sayed. Dynamic federated learning. *arXiv preprint arXiv:2002.08782*, 2020.

[71] Elsa Rizk, Stefan Vlaski, and Ali H Sayed. Dynamic federated learning. *arXiv preprint arXiv:2002.08782*, 2020.

[72] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 3–18. IEEE, 2017.

[73] David Silver et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[74] David Silver et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[75] Sebastian U Stich. Local sgd converges fast and communicates little. *arXiv preprint arXiv:1805.09767*, 2018.

[76] Zhibo Sun, Hong Zhang, Zixia Liu, Chen Xu, and Liqiang Wang. Migrating gis big data computing from hadoop to spark: an exemplary study using twitter. In *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*, pages 351–358. IEEE, 2016.

[77] James Supancic III and Deva Ramanan. Tracking as online decision-making: Learning a policy from streaming videos with reinforcement learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 322–331, 2017.

[78] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[79] Javid Taheri, Albert Y Zomaya, Pascal Bouvry, and Samee U Khan. Hopfield neural network for simultaneous job scheduling and data replication in grids. *Future Generation Computer Systems*, 29(8):1885–1900, 2013.

[80] Kristof Van Moffaert, Madalina M Drugan, and Ann Nowé. Scalarized multi-objective reinforcement learning: Novel design techniques. In *2013 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, pages 191–199. IEEE, 2013.

[81] Kristof Van Moffaert and Ann Nowé. Multi-objective reinforcement learning using sets of pareto dominating policies. *The Journal of Machine Learning Research*, 15(1):3483–3512, 2014.

[82] Vinod Kumar Vavilapalli et al. Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th annual Symposium on Cloud Computing*, page 5. ACM, 2013.

[83] Guolu Wang, Jungang Xu, and Ben He. A novel method for tuning configuration parameters of spark based on machine learning. In *High Performance Computing and Communications*, pages 586–593. IEEE, 2016.

[84] Jianyu Wang and Gauri Joshi. Cooperative sgd: A unified framework for the design and analysis of communication-efficient sgd algorithms. *arXiv preprint arXiv:1808.07576*, 2018.

[85] Lei Wang, Dongxiang Zhang, Lianli Gao, Jingkuan Song, Long Guo, and Heng Tao Shen. Mathdqn: Solving arithmetic word problems via deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[86] Liqiang Wang, Shiyong Lu, Xubo Fei, Artem Chebotko, H Victoria Bryant, and Jeffrey L Ram. Atomicity and provenance support for pipelined scientific workflows. *Future Generation Computer Systems*, 25(5):568–576, 2009.

[87] Shiqiang Wang, Tiffany Tuor, Theodoros Salonidis, Kin K Leung, Christian Makaya, Ting He, and Kevin Chan. Adaptive federated learning in resource constrained edge computing systems. *IEEE Journal on Selected Areas in Communications*, 37(6):1205–1221, 2019.

[88] Jun Wu, Xin Xu, Pengcheng Zhang, and Chunming Liu. A novel multi-agent reinforcement learning approach for job scheduling in grid computing. *Future Generation Computer Systems*, 27(5):430–439, 2011.

[89] Wentai Wu, Ligang He, Weiwei Lin, Rui Mao, Carsten Maple, and Stephen A Jarvis. Safa: a semi-asynchronous protocol for fast federated learning with low overhead. *IEEE Transactions on Computers*, 2020.

[90] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[91] Cong Xie, Sanmi Koyejo, and Indranil Gupta. Asynchronous federated optimization. *arXiv preprint arXiv:1903.03934*, 2019.

[92] Hung-chih Yang, Ali Dasdan, Ruey-Lung Hsiao, and D Stott Parker. Map-reduce-merge: simplified relational data processing on large clusters. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 1029–1040, 2007.

[93] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2):1–19, 2019.

[94] Nezih Yigitbasi, Theodore L Willke, Guangdeng Liao, and Dick Epema. Towards machine learning-based auto-tuning of mapreduce. In *2013 IEEE 21st International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 11–20. IEEE, 2013.

[95] Hao Yu, Rong Jin, and Sen Yang. On the linear speedup analysis of communication efficient momentum sgd for distributed non-convex optimization. *arXiv preprint arXiv:1905.03817*, 2019.

[96] Hong Zhang, Hai Huang, and Liqiang Wang. Mrapid: An efficient short job optimizer on hadoop. In *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 459–468. IEEE, 2017.

[97] Hong Zhang, Hai Huang, and Liqiang Wang. Meteor: Optimizing spark-on-yarn for short applications. *Future Generation Computer Systems*, 101:262–271, 2019.

[98] Hong Zhang, Zhibo Sun, Zixia Liu, Chen Xu, and Liqiang Wang. Dart: A geographic information system on hadoop. In *Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on*, pages 90–97. IEEE, 2015.

[99] Hong Zhang, Liqiang Wang, and Hai Huang. Smarth: Enabling multi-pipeline data transfer in hdfs. In *2014 43rd International Conference on Parallel Processing*, pages 30–39. IEEE, 2014.

[100] Fan Zhou and Guojing Cong. On the convergence properties of a *k*-step averaging stochastic gradient descent algorithm for nonconvex optimization. *arXiv preprint arXiv:1708.01012*, 2017.

[101] Ligeng Zhu and Song Han. Deep leakage from gradients. In *Federated Learning*, pages 17–31. Springer, 2020.