DECENTRALISED: SCHEDULING JOBS IN GEO-DISTRIBUTED DATA CENTERS

Shubhendra Pal Singhal

A THESIS

in

MSE in CIS

Presented to the Faculties of the University of Pennsylvania

in

Partial Fulfillment of the Requirements for the

Degree of Master of Science in Engineering

2022

Approved:____

Dr. Boon Thau Loo, Professor
Academic Advisor

Approved:____

Dr. Linh Thi Xuan Phan, Associate Professor

Approved:____

Dr. Swapneel Sheth, Practice Associate Professor
Graduate Program Chair MSE/CIS

ACKNOWLEDGEMENT

I would like to thank my outstanding advisors Boon Thau Loo and Linh Phan. Boon and Linh have been the perfect combination of advisors, and I feel very lucky to have the opportunity to work closely with both of them.

I have known Boon over more than a year, when I met him first in my Operating Systems course and since then, he has not just been advising me in research but also in life as a whole aspect. I could never wished for any better experience than being with him and working with him. He provided me a safe and supportive environment and a special mention to my dearest lab DSL, which changed my life from being dis-organised to a routine, which strengthened by discipline towards research and career goals ahead.

Linh has been a special advisor to me and I have been following her work since my initial days of joining University of Pennsylvania. Taking a distributed systems course with her, was my best decision as she provides some really challenging and interesting assignments to work on which are directly affiliated with the practical skill set required to design and develop such large-scale systems.

I would like to thank Peraton Labs for their constant support in engineering, which helped me gain professional development skills. Last but not the least I would like to thank my parents and friends who have constantly stood by my side and have shared their personal experiences of life.

ABSTRACT

DECENTRALISED: SCHEDULING JOBS IN GEO-DISTRIBUTED DATA CENTERS

Shubhendra Pal Singhal

In this thesis, we present the system design and implementation of experimental scheduler. Scheduler for cluster of datacenters have often being accompanied with its overheads and failure constraints. In addition, because of LAN network improvements, specific areas such as WAN bandwidth constraints, regulatory constraints by governments across, have been some of the major rising concerns.

Our contributions are as follows. First, we propose a decentralised system design for scheduling the jobs across datacenters in different clusters using WAN as their primary communication, which indeed is autonomous with respect to every server, as each server consists of both primary manager and secondary managers responsible for executing jobs and tasks respectively by controlling the resources: compute or input volume and data locality. There is no single master controller, responsible for distributing and successful execution of jobs across any cluster, rather every server is made responsible for the execution, therefore incorporating extra overheads in coordinating the jobs.

Second, we present a prototype with possible future extensions, in-order to help the research community experiment on specific optimizations such as task placement, monetary costs, data distribution and many more in scheduling across geo-distributed datacenters. The work for the scheduler is in progress and the basic version of the design is explained in this document whose possible extensions are additional modules that are currently in development.

Third, experimenting the scheduler (yet to be integrated with) with testbeds such as TPC, Yahoo production traces have indicated a strong urge to provide a common testing server platform alike Spark's scheduler, whose optimizations are no more an extension, but a part of the scheduler itself.

TABLE OF CONTENTS

CHAPTER 1

Introduction

Geo-Distributed data centers are increasingly common to derive useful information in large organisations. Nowadays, large organisations require datacenters either be compute or input volume storage, to process the client's requests. Geo-Distributed depends on two factors: first being that the clients are spread across multiple regions of the world, and second being that a single datacenter may be overloaded by the requests, which is why distributed enterprise settings are favoured.

Setting the clusters of datacenters have often dealt with numerous problems such as replication of data, data locality, large network transfers etc. which are often co-related to the general problems faced by any distributed system. One such major problem is scheduling of jobs in a geo-distributed datacenter. In section 2, we talk about various scheduling policies that have been explored with respect to network costs, monetary costs of data handling/compute resource handling and various government regulatory constraints, but all the policies/systems designed follow a common notion, of a master controller of a cluster requiring to unfold the jobs into its respective tasks using DAG, which manages the resources both computing and storage as a global parameter which then helps to decide the task placement and execution in a datacenter. Although master controller solves purposes like easy management using Zookeeper installed on the front to manage the servers and report the statistics to the master, ease of task placement and data distribution, we often tend to forget that if the master controller fails, then the impacts of such failure can be huge. Some of the major problems are: The whole cluster of the data centers would be down which makes all the respective servers to be not used, which is pure resource waste as the servers do not necessarily face the failure at the same time as the cluster manager fails, making the resource under-utilised. Second, the recovery of such a failure costs the master to run every task-chain or job-chain again, after the safe checkpoint, which in-return leads to additional overheads with respect to to maintaining consistency across servers, re-executing the whole job even if only the last part of the job was being executed, server management load on only one master which indeed leads to more probability of failures ahead and finally, recovery of the data from geo-located servers which might need approvals from government restrictions which would need to be executed again! These problems are often ignored by almost every system that we conducted our study in, and although master controller makes it at ease to program such system-designs, it's repercussions of failure are much worse and costlier to recover which can lead to non-functioning of an entire cluster of datacenter, which proves to be unfair for the servers which can still accept and respond to client's requests.

In this document, we have proposed a new system design which follows decentralised/autonomous system of master controller, where every server in the datacenter in a cluster acts as both the master manager and secondary manager, master doing the exact same job as the cluster manager in centralised design, and secondary held responsible for executing the individual tasks which are controlled by the primary manager of the another server! Henceforth, we try to study the overheads of such design by building an experimental scheduler setup following the de-centralised design and compare it with the most efficient centralised scheduler as proven by some researchers following network locality, data locality, hybrid modes of locality and monetary costs as the different parameters in consideration.

This in addition also helps us bring forward a new design for scheduling in a de-centralised/co-operative fashion, which can later be integrated as a part of Spark scheduler which does a refined job of following the RDDs for map-reduce framework. This integration in future will help us improve the scheduler optimizations and henceforth, change the cluster failures to almost minimal recovery costs given the feature that the scheduler is application agnostic.

CHAPTER 2

Related Work

2.1. Remote Procedure Calls

Remote Procedure calls are an important part of any distributed system design, where every stack of the RPC communication starting from hardware to application layer is shown in the figure 2.1.

Developing n experimental scheduler, requires us to first study what stack components would be suitable. In general, most of the scheduling algorithms have been compared to Spark's scheduler, which uses Apache Thrift as RPC communication. But, we need to address the scheduler from the point of view, which has the support for multiple components, which can indeed allow more scope of testing with respect to available closed source software as well. We plan to support the following features explained with reference to stack mentioned in 2.1.

- From the client-server model, we have decided to choose the multi threaded pool where every server has one thread corresponding to primary master and other threads corresponding to secondary managers as per the request.

- We use JSON for reading all the server configurations and system setup but we also at the same time, support Protocol Buffer, as the means of RPC communication. Extensions to using JSON would be a future extension which supports the communication using the communication flag as a parameter given for server-setup.

- Current scheduler does not use any well-defined protocols like HTTP2/1, but rather uses own-defined protocol, which is a simple communication protocol specifying the parameters and the type of request with TCP support. Extension to security layers can be beneficial for precise measurements for transfer of data, but is kept for future considerations.

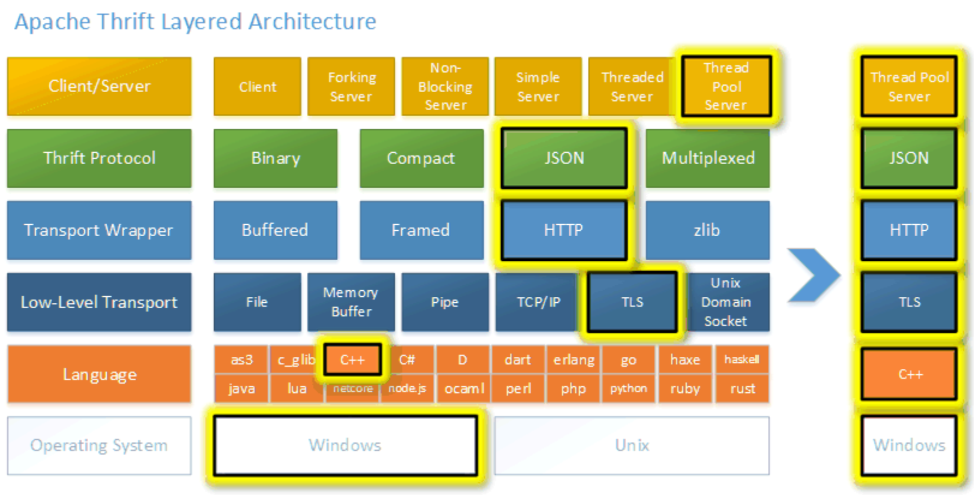


Figure 2.1: Apache Thrift, [Sourced: https://github.com/apache/thrift]

- For language support, C++ is used. The scheduler has been tested on Linux Ubuntu 20.04LTS. Extensions to java is in consideration but currently not in the immediate future plans for the scheduler.

2.2. Spark

Map-Reduce is a concept which helps the application to run in parallel run-time environments and then reduce it to produce the output. But one of the major problems faced was the size of map-tasks which has been extensively larger than the RAM size available, therefore Spark came up with the RDD's Zaharia et al. (2012) which are small metadata sets for the map-tasks which often contains the details of the map-tasks essentially required at the time of execution, which significantly reduces the RAM used by the map-task as the whole map-task is not required to be loaded and specific application mapping, can be setup in-relation to the RDD's required.

2.3. Centralised Master Scheduling

There are various studies conducted which are often categorised into two different groups: first being offline where all the job's characteristics are available and the scheduler performs the scheduling operation on a batch of jobs disregarding any new job arriving in that interval of scheduling, second being online, where the job's characteristics are estimated by

the models of RL, previous similar process history on the basis of resources used, process type etc. which then scheduler takes into account and delivers the next job that is ready to execute in the queue. The priority order for the rest of the jobs is calculated by considering all the other jobs arriving at a latter time as well, which gives us the flexibility to consider the job's according to their arrival times, and not on a fixed set of jobs.
The offline algorithms often include execution time, compute and input volume resources as the parameters along-with the recent trends of considering WAN bandwidth and network transfer as other parameter, which has been demonstrated in Hung et al. (2015).

One such demonstration of a recent offline algorithm which concentrates on resources used in perspective of geo-distributed has been explained below.
With growing data volumes generated and stored across geo-distributed datacenters, it is becoming increasingly inefficient to aggregate all data required for computation at a single datacenter. Instead, a recent trend is to distribute computation to take advantage of data locality, thus reducing the resource (e.g., bandwidth) costs while improving performance. In this trend, new challenges are emerging in job scheduling, which requires coordination among the datacenters as each job runs across geo-distributed sites. Natural SRPT(Shortest Remaining Processing Time)-based extensions with re-ordering improves the existing offline scheduling mechanism. Improvements are generally measured as either the makespan of jobs which is the worst execution time of all the jobs considered in a set, and average job response time (JCT). These parameters have been considered in addition to bandwidth costs, network transfers and data-locality as parameters.

Online algorithms considering adaptive feedback for determining the job's characteristics have taken an interesting turn by not only including the previous work of obtaining the parameters from process's/job's history but also identifying the hyper parameters using RL model. One such demonstration of a recent online algorithm which concentrates on all the

parameters such as WAN bandwidth, heterogeneity of network resources and many more has been considered as knobs which are included as optimization problems whose solvers output the required datacenter allocations and task placements.
Tetrium Hung et al. (2018), a system for multi-resource allocation in geo- distributed clusters, jointly considers both compute and network resources for task placement and job scheduling.

2.4. De-centralised Scheduling

Recent survey conducted on advancements in geo-distributed data centers Bergui (2021), suggests that there is a significant lack in the system-designs in this area. Regardless, the system analysed HOUTU, Zhang et al. (2018), has provided us a good insight on how the decentralised/co-operative scheduling works. Although assumptions such as no consideration for efficient splitting of data distribution across datacenters, task placement, a completely connected system where every datacenter is connected to each other directly via a up/downlink has to be worked on, but the major key takeaways are the following:

- Every data-center/server should have its own primary manager and secondary manger to work on jobs and tasks respectively.
- WAN bandwidth links cannot be assumed to be constant, rather a similar instance on AWS should be considered for benchmark purposes.
- Monetary costs, makespan and JCT should be measured as the performance metrics, along-with the additional scheduling overheads.

These system-designs often face one major problem, i.e. common grounds of testing the scheduler. The lack of frameworks that support decentralised architecture and multi-clusters often pose a great challenge of testing beds which can induce new concepts of optimization for communication, network transfers, task placement, data distribution and scheduling jobs which motivates us to build a fine grained server-client based scheduler for the purpose of detecting these possible optimizations (which is the beginning step for experimenting with new system-designs for adoptable decentralised scheduling). We believe that a decentralised geo-distributed big data system can offer great flexibility in deployment as it provides autonomous geo-distributed clusters that can coordinate for geo-distributed jobs. Such system can avoid a general outrage of the whole system in the case of DC failure. Moreover, a decentralised system combined with a security/authentication mechanism can deal with the regulatory constraints and restrictions.

Some systems integrate the scheduler with Spark in-order to make map-reduce of the jobs automated, whereas some systems lack the feature of consideration of multiple staged tasks, task parallelism. Some system-designs operate on comparisons on older versions of Spark's scheduler which have been vastly improved both application agnostic and application specific- using efficient graph partitioning mechanisms, task placement strategies, network transfers etc. In this document, we therefore come up with the experimental scheduler which enables the user to test their respective scheduling algorithms, using different types of serialisation/de-serialisation techniques of RPC, different communication modes- server-reply; publish-subscribe, task-parameter estimation modules. Providing a common platform for analysis helps the researchers to analyse the efficiency of their new algorithm which runs against all the provided algorithms, being application agnostic as of today. More specific details are listed in the section below.

CHAPTER 3

Experimental Scheduler: Key Design

3.1. Terminologies

The system runs for uni-processor scheduling perspective with basic consideration of server setup using server_config.json which helps us initialize the datacenter servers, assuming every datacenter has one server.

- The job is defined as a DAG of tasks which would either include characteristics extracted by optimizer or given as an input. Making a separate module for optimiser is left to the user which can in-turn generate the json file for jobs which acts as a common repository for all servers for which they can download the job descriptions primarily including the tasks. If the job descriptions need to be sent via network, there is always an extension of adding the monetary costs related to the bandwidth of the link depending on the jobs data size which can be extracted from the protocol buffer.

- Task is defined as the smallest entity which is defined as the smallest unit obtained from Map-Reduce framework for the application or can be manually divided by the user. The experimental scheduler in future would have support for Spark which would avoid user to manually enter the tasks. The scheduler currently takes any script whether be it python, bash or any other which can be executed on shell using execvp using the data locally available. If the data transfer over network is in consideration, we can tune the system to add the data transfer costs, which can primarily be included by the user for better experimental results, i.e. creating similar simulation conditions, making this knob open for calculations.

- Queues for scheduler are separate for jobs and tasks, because the primary manager would be responsible for scheduling the jobs that it receives, whereas the secondary manager would be responsible for scheduling individual tasks. Different scheduling

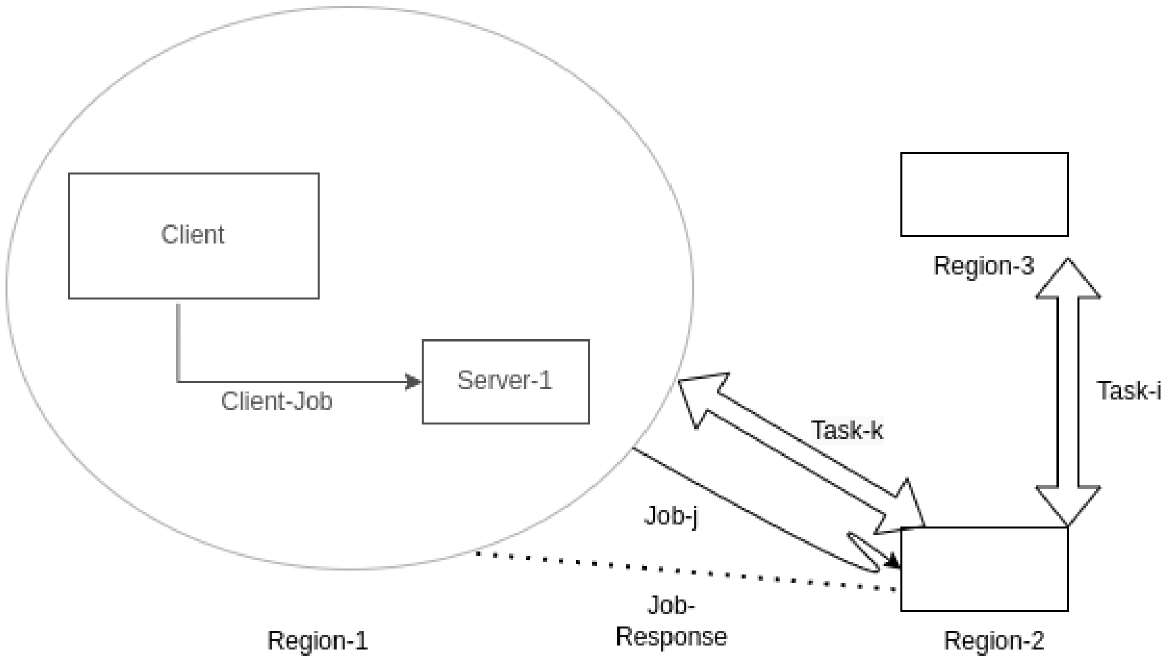


Figure 3.1: System-Design: Broader View

algorithms are available for interpretation which can be extended by adding more modules in scheduler which has to be consistent with the task's and job's characteristics. The program is compatible to analyse the parameters required, if not found compatible, will result in faults shown in the logs.

- Logging for individual data-center/server has been enabled which helps us analyse the timestamps local to the server for in-depth analysis of every task execution, task arrival and so on, which are represented by the struct which contains all the information which can either be extracted from logging or can be extracted from the main/local-region server which received the request for the job first from the client.

- Job-ids and corresponding tasks-ids are matched with what user provides but the constraint is that every job in the json file provided or generated would have to be unique in addition to the hash with timestamp in-order to uniquely identify the request with respect to job or task.

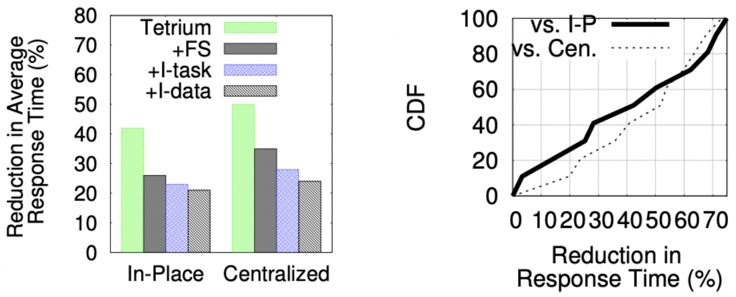
3.2. System Design: De-centralised

- The client sends the request as communication type, "Client:Job" which eventually is sent to the local servers specific to the region(allocated as regions in geo-distributed regions) which would then forward the request to the datacenter/server which is responsible for executing the job.

- The local server receives the request which would be forwarding the request to the desired data-center which will also map the unique id to the job which would be stored in the job_map. These ids are therefore, forwarded as unique_job_id which is separate from the user-ids provided because user-ids are primarily responsible for reading the job-description from the client/user, whereas the unique id would be responsible for forwarding the response back from the server/client back.

- After the desired server receives the required request as "Server:Job" from the regional server geo-located with client, it sets the arrival time of the job, and then puts the job in the scheduler queue. Timestamp for arrival-time is important is because the waiting time, as well as the execution time can be measured which is unaffected the clocks of any server, because it is eventually the difference of timestamps primarily depending on the clock-cycles of the CPU which we have taken into consideration, when measuring the time, which is local to any server, thereby making a constant in SI unit of seconds (measured as double). In addition, if this was the request which was the first index of the queue, it would set the schedulerFlag to true, indicating the scheduler to run the following algorithm. In this way, the scheduler ends running only when all the jobs in the queue are served.

- The next step is to highlight the following fact that the job_queue serves the job first in-accordance with the algorithm being followed, which means that the job request cannot be expected for an instant reply. So, at the timestamp, when the schedulerFlag is set, it would recognise the job which should be ready to execute, which then follows



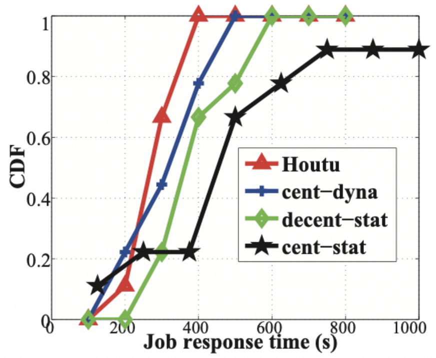
(a) Average Response Time (b) CDF of Response Time

Figure 3.2: Optimizations in Centralised Scheduling: Tetrium

the DAG of tasks. In-order to follow the DAG, we follow BFS and allocate the tasks to respective data-center. When any one level of BFS is traversed and received the response for all nodes in that level, the time is measured as the time instant of the last node's response received minus the issue time of the task. This follows until the last node of the DAG has been served. The requests are sent as "Task:Reply" and the response is collected as "Task:Response". The same mechanism of creating a new task id which is unique for map and task-id responsible for reading the tasks from json file follows the same as jobs explained above.

- An additional flag which has been considered as a part of the scheduler is the network topology. This is the future extension which would need to follow the different links to reach a specific data-center which would add an additional overhead of recognising the least network transfer time link, which has to be weighed over two separate case assumptions: first considering all data-centers to be connected directly and second, routing through any route without calculating the best route possible!

All the work has been made available by the author on Singhal (2022).



(a) CDF of job response time.

	Average JRT	Makespan
Houtu	290 s	387 s
cent-dyna	295 s	417 s
decent-stat	377 s	561 s
cent-stat	488 s	1109 s

(b) average job response time and makespan.

Figure 3.3: Decentralised: Cooperative Scheduling

3.3. Expected Results

Centralised scheduling has been improved for specific applications per say, by almost 20-40% reduction in time as shown in 3.2, whereas we can clearly see Zhang et al. (2018), making only a small reduction of 5-20% as shown in 3.3 which is incredible, but considering the assumptions that have been made and no accountability of those overheads have been posed, there is a greater need to improve the decentralised mechanism of scheduling as well as standardize the comparison by building a scheduler alike Spark, including the optimizations and improvements suggested over the years, rather than treating these as extensions as most of these are not application specific as a clear case in Hung et al. (2015).

CHAPTER 4

Conclusion and Future Work

Problem of designing new co-operative/decentralised scheduling problems keeping the research done so far for centralised scheduling, requires a common testing scheduler which can be used as a autonomous test beds for any new policy implemented being, it optimization in task placement, data distribution, network-data locality issues and many more.

- Integration with Spark: As of this moment, the scheduler has the tasks and jobs stored as independent DAGs, which are eventually obtained from the map-reduce framework. We assert that the task generation from map-reduce is application and user-dependent, so integration with Spark, requires us to join the mechanism of writing the task info in the json file corresponding to task or job.

- Support for gRPC: Supporting a bidirectional stream of data can be helpful for schedulers dealing with video/graph applications which has been efficiently optimised in gRPC, especially for streaming data. Multiple options such as RPC chaining can be brought forward, which would be pretty straight forward as the jobs would then be integrated as DAGs of jobs.

- Testing beds with TPC workloads: There are various popular production traces like Yahoo, TPC-AI, TPC-H benchmarks when run as jobs on the scheduler, would produce precise results with respect to waiting times, execution times and network transfer times. There is an option to use the characteristics of jobs as well which can be tested on with accuracy comparing it with the optimizers used to obtain the characteristics.

- Implementing the job-characteristics module as either optimization solution or extraction from any well-defined benchmark would be a future extension of the scheduler, which can be provided as a support of example modules derived from Hung et al. (2018).

- Consideration of data distribution across servers, being application dependent would be an additional parameter which needs a separate module all together, but as of now, for any timing measurements, equivalent measurement using bandwidth and network loads are precisely managed.

BIBLIOGRAPHY

Najah S. Nikolov N.S. Bergui, M. A survey on bandwidth-aware geo-distributed frameworks for big-data analytics. 2021. doi: 10.1186/s40537-021-00427-9. URL https://journalofbigdata.springeropen.com/articles/10.1186/s40537-021-00427-9#citeas.

Chien-Chun Hung, Leana Golubchik, and Minlan Yu. Scheduling jobs across geo-distributed datacenters. In *Proceedings of the Sixth ACM Symposium on Cloud Computing*, SoCC '15, page 111–124, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450336512. doi: 10.1145/2806777.2806780. URL https://doi.org/10.1145/2806777.2806780.

Chien-Chun Hung, Ganesh Ananthanarayanan, Leana Golubchik, Minlan Yu, and Mingyang Zhang. Wide-area analytics with multiple resources. In *Proceedings of the Thirteenth EuroSys Conference*, EuroSys '18, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450355841. doi: 10.1145/3190508.3190528. URL https://doi.org/10.1145/3190508.3190528.

Shubhendra Pal Singhal. Scheduling in geo-distributed datacenter, 2022. URL https://github.com/singhalshubh/Linh-Boon_RA.

Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauly, Michael J. Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A Fault-Tolerant abstraction for In-Memory cluster computing. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 15–28, San Jose, CA, April 2012. USENIX Association. ISBN 978-931971-92-8. URL https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/zaharia.

Xiaoda Zhang, Zhuzhong Qian, Sheng Zhang, Yize Li, Xiangbo Li, Xiaoliang Wang, and Sanglu Lu. Towards reliable (and efficient) job executions in a practical geo-distributed data analytics system, 2018. URL https://arxiv.org/abs/1802.00245.