Efficient Task Placement for Geo-Distributed Data Processing

Qingyuan Wang
Supervisor: A/P Yong Meng Teo
Department of Computer Science
National University of Singapore

April 2018

Abstract

Low latency analytics on geographically distributed datasets (across datacenters, edge clusters) is an upcoming and increasingly important challenge. The dominant approach of aggregating all the data to a single datacenter significantly inflates the timeliness of analytics. At the same time, running queries over geo-distributed inputs using the current intra-DC analytics frameworks also leads to high query response times because these frameworks cannot cope with the relatively low and variable capacity of WAN links.

In order to explore the impact of some aspects in geo-distributed data processing, I develop an analytical modeling approach to determine the total completion time for a single job on a geo-distributed clusters. I validate the accuracy of the proposed model on a geo-distributed Apache Spark cluster spanning multiple Amazon AWS cloud regions. Additionally, I simulate jobs using the model, explore and discuss factors that impact the job performance in geo-distributed data processing.

1 Introduction

With big data explosion and the rise of Internet-of-Things during recent years, efficiently processing geographically distributed data is becoming increasingly important. Large scale corporations are deploying datacenters and clusters globally on the "edge" to provide their users low latency access to their services. For instance, Microsoft and Google have scores of datacenters (DCs) [1, 2] providing services to various organizations. The services deployed on these geo-distributed sites generate a large amount of data continuously, such as, user activity, session logs, server monitoring logs, and performance counters among others. In addition, there are some geo-distributed applications that process and analyze this large volume of geo-distributed data to obtain the final result. Because these analytics jobs usually support the real-time decisions and on-line predictions, minimizing response time are essential. However, these face the unique challenges of wide area network (WAN) bandwidth limits, legislative and regulatory constraints, unreliable runtime environment, and even monetary costs.

Distributed data processing is a well-studied topic in computer science. The development of distributed processing frameworks, such as Hadoop and Spark, have led to scalable, fault tolerant and efficient real-time and batch data processing. However, all these popular systems have a major drawback in terms of locally distributed computations. When input data is located across multiple datacenters, conventional approach is to aggregate all the data to a single datacenter before processing. Naturally, transferring huge amounts of data across datacenters may result in substantial network traffic and increase job completion time.

The natural alternative to this approach is to leave data in-place and use unmodified, intra-DC analytics framework (such as Hadoop or Spark) across the collection of sites. However, such distributed frameworks are designed for non-geo-distributed clusters, thus lack consideration to some of the major characteristics in geo-distributed clusters and job execution could be dramatically inefficient. For instance, in a local cluster, the intra node network is often homogeneous and with very high consistent bandwidth, while geo-distributed clusters have heterogeneous network links connecting nodes with fluctuating network bandwidths. Therefore, state-of-the-art distributed processing frameworks need to be extended for geo-distributed data clusters, while giving due concern for new challenges in the geo-distributed network and data.

One of the major challenges in geo-distributed processing systems is determining the task placement among many sites. There are existing research that have attempted different heuristic based optimization approaches to improve task placement. However, they lack concern to the heterogeneous nature of the network connecting the sites, do not consider applications with complex data flow having multiple execution stages, of take too long to get the task placement due to compute expensive optimization algorithm making them less applicable in realistic geo-distributed processing systems.

Given an application with jobs having multiple execution stages represented by a Directed Acyclic Graph (DAG), I develop an analytical modeling approach to determine the total completion time for a single job on a geo-distributed cluster. I assume that the data is initially non-evenly distributed across the geo-distributed sites and the network connecting these sites have heterogeneous links with different link bandwidths. Take Apache Spark as an example, I validate our model on a geo-distributed Spark cluster spanning multiple Amazon AWS cloud regions. Using our model, I present useful insights into the impact of different aspects in geo-distributed data processing on job execution time.

2 Related Work

With increasingly large volumes of data generated globally and stored in geo-distributed datacenters, people are performing an increasing amount of research attention to processing data throughout multiple datacenters. The main ideas are similar: making query execution (specifically, data and task placement) WAN-aware. Based on their objectives, existing efforts can be roughly divided into two categories: reducing the amount of inter-datacenter network traffic to save operation costs and reducing the job completion time to improve application performance.

Reducing the amount of traffic among different datacenters is proposed in [3, 4]. In[3], they design an integer programming problem for optimizing the query execution plan and the data replication strategy to reduce the bandwidth costs. As they assume each datacenter has limitless storage, they aggressively cache the results of prior queries to reduce the data transfers of subsequent queries. In Pixida [4], they propose a new way to aggregate the tasks in the original Directed Acyclic Graph to make the Directed Acyclic Graph simpler. After that, they propose a new generalized min-k-cut algorithm to divide the simplified Directed Acyclic Graph into several parts for execution, and each part would be executed in one datacenter. However, these solutions only address bandwidth cost without carefully considering the job completion time.

As a representative work in the second category, Iridium [5] proposed an online heuristic to place both data and tasks across datacenters. Unfortunately, the model in Iridium is based on a few assumptions that are not realistic. First, they assume that sites are connected using a network with congestion-free core and the network bottlenecks only exist in the up/down links of sites. Second, their work can only be applied to simple 1- or 2-stage queries without distinguishing different dataset. Third, they assume that sites have relatively abundant compute and capacity and that I/O and CPU operations of tasks have zero duration.

Flutter [6] and Chen [7] removed the assumption that network core is congestion-free and used a pair-wise connectivity model. What's more, rather than scheduling all the tasks together, Flutter [6] schedules ready tasks stage by stage in an online fasion. They formulated a lexicographical minimization problem of task assignment for a single stage of one job, and obtained its optimal solution. Chen [7] take competition for resources among concurrent jobs into consideration. Using a similar theoretical foundation as [6], they designed and implemented a new optimal scheduler to assign tasks across geo-datacenters, in order to better satisfy job requirements with max-min fairness achieved across their job completion times.

Clarinet [8] is designed for SQL queries. They also abstract the WAN as a logical full mesh with fixed bandwidth links. It feeds the bandwidth information to the query plan optimizer to generate better query plans for queries over geo-distributed data. SWAG [9] adjusts the order of jobs across data centers to reduce job completion times, which is orthogonal to my work.

None of existing work model task processing time and most of them only optimize data transfer time in their task placement algorithm. Nevertheless, task processing usually takes up a significant proportion of job completion time according to observations. And experiments show that the improvement of time efficiency is limited, if the task placement strategy doesn't take task processing time into consideration.

Table 1: Summary of Related Work

Works	Processing type	Data placement	Task placement	Modeling processing time	Connectivity	Multiple jobs
Iridium [5]	general	✓	once per job	-	network core	-
Clarinet [8]	SQL	-	once per job	-	Pair-wise	✓
Flutter [6]	general	-	stage by stage	-	Pair-wise	-
Chen, et al. [7]	general	-	stage by stage	-	Pair-wise	✓
Our work	general	-	stage by stage	✓	Pair-wise	-

A comparison of algorithms aimed at time-efficient geo-distributed big-data processing based on several aspect such as data processing type, approach, task placement times and considerations is given in Table 1.

In the past few years, performance modeling in MapReduce or Spark environments has also received much attention, and different approach [10, 11, 12, 13] were offered for predicting performance of MapReduce or Spark jobs. These works focus on MapReduce or Spark performance in a single cluster, while I focus on data processing jobs in a geo-distributed cluster.

3 Approach

3.1 Overview

I consider the geo-distributed data processing framework to logically span all the sites. The sites are inter-connected by WAN. I abstract the WAN as a logical full mesh, namely pair-wise connectivity between sites. On popular cloud platforms (e.g., Amazon EC2 and Google Could), inter-datacenter wide-area networks are provided as a shared service, where user-generated flows will compete with millions of other flows. As a result, each inter-datacenter TCP flow will get a fair share of the link capacity [7]. Additionally, there could be significant heterogeneity in the WAN bandwidths due to widely different link capacities and other traffic sharing the links. Different sites have different number of processing units and processing units in different sites have different compute throughput. Finally, I assume the sites have relatively abundant storage capacity.

Data can be generated and stored on any site and as such, a dataset (such as "user activity log for application X") could be distributed across many sites. To facilitate parallel processing, dataset is divided into *partitions.* So, in geo-distributed environment, different partitions of one dataset could be distributed cross many sites.

For a geo-distributed data processing job, a logically centralized *driver* converts the user's script into a DAG, where vertices represent *stages* and the directed edges represent the *dependency of stages.* A stage is a physical unit of execution. And it has a set of parallel *tasks.* The same task is done over different partitions of data. The number of tasks within each stage is determined based on the size of the input data and configuration settings of the program. And stages are separated at shuffle boundaries. Shuffle boundaries introduce a barrier where stages/tasks must wait for the previous stage to finish before they fetch intermediate data. Shuffle is where "all-to-all" shuffle occurs. And data transfer across sites occurs mainly when one stage shuffle read intermediate data from previous stage.

In a geo-distributed setup, the main aspect dictating completion time of data processing job is efficient transfer of intermediate data that necessarily has to go across sites (e.g., all-to-all communication patterns) and balanced processing workload among workers.

3.2 Model

For a given job, it can be represented a DAG as follows:

$$G = (V, E) \tag{1}$$

where a vertex v denotes a stage v in the job. A directed edge (u, v) means stage u has dependency on stage v. And v_0 is the final stage.

The set of parent stages of stage v, namely set of child vertices of vertex v is

$$P(v) = \{u|(v, u) \in E\}, \forall v \in V \tag{2}$$

First, I represent the job completion time, T, as the sum of the end time of final stage plus the job startup time and the job cleanup time as follows:

$$T = Startup + t_{v_0} + Cleanup, \forall v \in V \tag{3}$$

Symbol	**Meaning**
D	the set of sites
B_{kj}	WAN bandwidth from site i to site j
U_j	number of processing units on site j
C_j	throughput of each processing unit on site j
G	the DAG of stages
V	a set of vertices, each one corresponds to a stage in the job
E	a set of directed edges, each one corresponds to a dependency
P_v	a set of parent stages of stage v
N_v	the set of tasks in stage v
A_v	the set of data sources in stage v
I_{vija}	for task i in stage v, size of input data of source a on site j
S_{vi}	for task i in stage v, size of input data
x_{vij}	$x_{vij} = 1$ indicates the assignment of task i in stage v to site j; otherwise $x_{vij} = 0$
M_{vj}	for stage v, number of tasks assigned to site j
T	job completion time
t_v	end time of stage v
τ_{vj}	for stage v, the total task execution time in site j
τ_{vij}	task execution time of task i assigned to site j in stage v

Next, each stage can not start until all the parent stages of it have completed. And each stage contains multiple tasks. Tasks assigned to different sites can be executed in parallel and the stage execution time is determined by the slowest task execution time among all the sites. I denote the set of sites by $D = \{1...d\}$. Therefore, the end time of a particular stage v can be calculated as the maximum of tasks execution time among all the sites plus the stage startup time, the stage cleanup time and time of waiting for parent stages as follows:

$$t_v = \max_{u \in P(v)} t_u + Startup + \max_{j \in D} \tau_j + Cleanup, \forall v \in V \tag{4}$$

Task placement constraints: I denote the set of tasks in a stage by $N_v = \{1...n\}$. So, the number of tasks is n. x_{vij} denotes whether i-th task in stage v will be assigned to the j-th data center. $x_{vij} = 1$ indicates the assignment of the i-th task to j-th data center; otherwise $x_{vij} = 0$.

$$\sum_{j=1}^{d} x_{vij} = 1, \forall v \in V, \forall i \in N \tag{5}$$

$$x_{vij} \in \{0, 1\}, \forall v \in V, \forall i \in N, \forall j \in D \tag{6}$$

Therefore, the number of tasks assigned to site j is

$$M_{vj} = \sum_{i=1}^{n} x_{vij}, \forall v \in V, \forall j \in D \tag{7}$$

Then, tasks executed in one site from one stage are independent. However, one processing unit (one CPU core, by default) executes one task at a time. So the number of processing units is the number of tasks can be executed in parallel in one site. If the number of tasks in stage v assigned to site v, M_{vj}, is not more than the number of processing units in this site, U_j, the total task execution time of site j is

$$\tau_{vj} = \max_{i \in N}\{x_{vij} \cdot \tau_{vij}\}, \forall v \in V_n, \forall j \in D, M_{vj} \leq U_j \tag{8}$$

If the number of tasks in stage v assigned to site j, M_{vj}, exceeds the number of processing units in this site, U_j, the total task execution time of site j is

$$\tau_{vj} = \frac{\sum_{i=1}^{N}\{x_{vij} \cdot \tau_{vij}\}}{U_j}, \forall v \in V_n, \forall j \in D, M_{vj} > U_j \tag{9}$$

Finally, I denote the set of data sources in a stage by $A_v = \{1...a\}$. So, the number of sources is a. For task i in stage v, size of input data is

$$S_{vi} = \sum_{k=1}^{d} \sum_{b=1}^{a} I_{vikb}, \forall v \in V, \forall i \in N \tag{10}$$

where I_{vikb} is the size of input data of source b from site k.

A task can be broken up as scheduler delay + task deserialization time + shuffle read time (optional) + executor computing time + shuffle write time (optional) + result serialization time + getting result time (optional). I compute them if task i is assigned on site j in equation 11. Shuffle read time is the time to fetch data from all the sites containing input data, which is determined by slowest data transfer among all the input sites. The sum of executor computing time, shuffle write time, result serialization time and getting result time is determined by total input data size and throughput of processing unit. Other parts can be regarded as a constant. Since each part can not start until preceding part ends, task execution time can be modelled as follows:

$$\tau_{vij} = Startup + \max_{k \in D}\left\{\sum_{b=1}^{A} \frac{I_{vikb}}{B_{kj}}\right\} + \frac{S_{vi}}{C_j}, \forall v \in V_n, \forall i \in N, \forall j \in D \tag{11}$$

where the WAN bandwidth from site k to j, B_{kj}, can be measured and is stable for a few minutes [5]. And C_j is throughput of each processing unit on site j is different in terms of many factors, such as memory size, application and so on.

Given the model, the task placement problem can be formulated as:

$$\begin{aligned} &\text{minimize} && T \\ &\text{subject to} && (1),(2),(3),(5),(6),(7),(8),(9),(10),(11). \end{aligned}$$

This formulation is not a linear program and not efficient to compute the optimum task placement. So I just use a brute force way to figure out the optimum result and do analysis.

4 Evaluation

In this section, I evaluate the accuracy of the model using a set of applications and a Spark cluster across sites in different regions. Then, by performing analysis of job profiles and application completion times, I reveal factors that impact the Spark job performance in a geo-distributed environment.

4.1 Experimental Setup

I first describe the testbed I used in my experiments, and then briefly introduce the applications, baselines, and metrics used throughout the evaluations.

Table 2: Available bandwidths between VMs across geodistributed data centers on EC2 as of April. 2018 (Mbps)

bandwidth	Oregon	Sao	Flankfurt	Singapore
Oregon	-	15.5	17.7	17.7
Sao	16.2	-	14.3	8.38
Flankfurt	18.4	13.6	-	17.7
Singapore	18.4	7.92	17.4	-

Testbed: I conduct experiments on Spark in standalone mode using EC2. My cluster on Amazon EC2 consists of instances across four regions (Oregon, Sao Paulo, Frankfurt and Singapore), two instances per region. All the instances are m4.large and each instance has two vCPUs, 8 GB of main memory and 60 GB of general purpose SSD (GP2). The bandwidths between VMs across regions can be found in Table 2.

The distributed file system is the Hadoop Distributed File System (HDFS). The block size in HDFS is 128MB, and the number of replications is 1. I use one instance in Oregon as the master node for both HDFS and Spark. All the other nodes are served as data nodes and worker nodes.

Workloads: I deploy WordCount on Spark. It is common workloads used in previous work. WordCount: WordCount calculates the frequency of every single word appearing in a single file or a batch of files. It would first calculate the frequency of words in each partition, and then aggregate the results in the previous step to get the result. I choose WordCount because it is a fundamental application in distributed data processing and it can be used to process the real-world data traces such as Wikipedia dump.

Baselines: I compare my optimal task placement to four baselines:

1. "centralized" aggregation of data at a main DC.
2. place tasks equally among all the sites.
3. balance the transfer times among the WAN links stage by stage, without considering processing time, which is similar to strategy adopted in [6].
4. balance the processing times among all the sites, without considering data transfer time stage by stage, which is similar to strategy adopted in most cluster processing framework.

Metrics: My primary metric is job completion time and stage completion time.

4.2 Validation

To evaluate the accuracy of my analytical model, first, I collect all the parameters in my model from the web UI that are generated by the Apache Spark platform to record execution profiles and performance metrics that are directly obtained from the Spark event listeners in the Apache Spark program. Then, I do more experiments using different input data in real world and compare the model output with real system output. I calculate the prediction accuracy for each stage and the whole job as well.

Table 3: Validation of Model: Comparison With Real System Data

Number of sites	Measurements	Map stage	Reduce stage	Whole job
2	real world time (s)	114	21	140
	model output (s)	135.4	19.9	161
	accuracy	81.2%	94.8%	85.0%
3	real world time (s)	166	60	231
	model output (s)	167.5	60.3	233.8
	accuracy	99.1%	99.5%	98.8%

Table 3 show the accuracy for time prediction for different experiment setup. As can be seen in the figure, sometimes the model achieves great prediction accuracy but not all the time. This may be due to the fact that the same applications and same setup can result in varying stage completion time and job completion time, which is observed in experiments.

4.3 Analysis

I use my model to simulate some jobs to analyse the impact of some parameters. I change one of the parameters and fix all the others.

4.3.1 Different Number of Sites

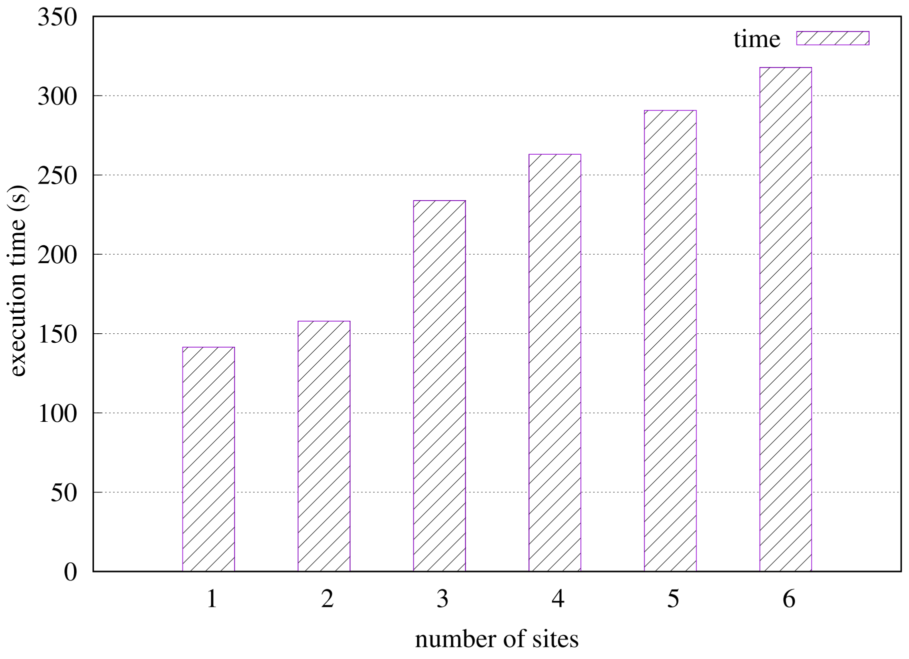

Figure 1: Job completion time varying with number of sites

Firstly, I explore the job completion time varying with the number of sites. Each site contains the same number of processing units and the same size of input data. In other words, when the number of sites increase, the total number of processing units and input data size increase linearly. The number of map tasks change linearly to take advantage of the increase in number of processing units. However, the number of reduce tasks does not increase, so the reduce job computing time increase. In addition, the framework need more time on data transfer because data need to be transfered increase linearly. Figure 1 shows the impact of number of sites. We can learn from it that job completion time increase when the input data size increase, but much slower.

4.3.2 Different Task Placement Strategies

Then, I explore the impact of task placement. I use an application process data distributed in 5 sites, 127 MB in each site. But the bandwidth and computing resource are heterogeneous. I compare several task placement strategies. The final one is using a brute force way to figure out the optimum task placement stage by stage. I plot the job completion time of all five strategy in Figure 2. As we can see,

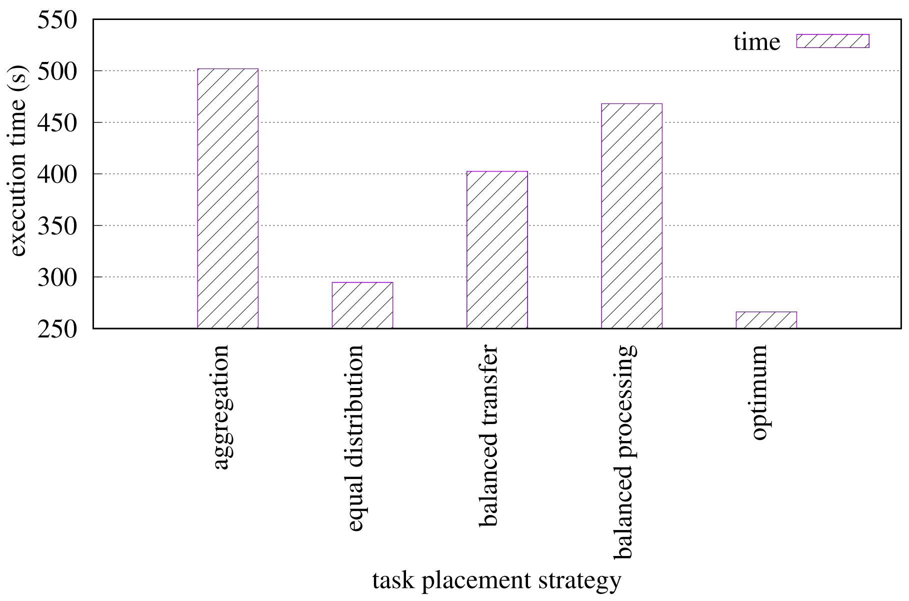

Figure 2: Job completion time varying with number of sites

Aggregation is very costly in terms of time, considering either data transfer time or processing time can achieve a better result, reducing job completion time by 19.8% and 6.8%, respectively. But both of them have a significant difference between the optimum strategy, which reduces job completion time by 47.0% compared to the aggregation strategy. In addition, placing tasks equally across the sites also achieve a good result (reduction by 20.0%) due to the input data in distributed equally in the sites. And this may not be a good idea in other cases.

5 Conclusion

In this work, I consider the problem of running data processing jobs over data gathered and stored at multiple sites inter-connected by heterogeneous WAN links. I have focused on the design and evaluation of an analytical model for predicting job completion time in geo-distributed environments. I have assessed the accuracy of the model using word count application. The accuracy of predicted completion time is above 80% of the measured ones in a real cluster.

In simulation experiments, I have evaluated some factors that impact the job performance in geo-distributed data processing. I found that when the amount of data increases according to the number of sites, the time of data processing increases sublinearly. And a good task placement strategy can improve the performance of geo-distributed data processing. However, the improvement of time efficiency is limited if the task placement strategy doesn't take either task processing time or data transfer time into consideration.

References

[1] Google. (2018, Apr) Google Datacenter Locations. [Online]. Available: http://www.google.com/about/datacenters/inside/locations/

[2] Microsoft. (2018, Apr) [2] Microsoft Datacenters. [Online]. Available: http://www.microsoft.com/en-us/server-cloud/cloud-os/global-datacenters.aspx/

[3] A. Vulimiri, C. Curino, P. B. Godfrey, T. Jungblut, J. Padhye, and G. Varghese, "Global analytics in the face of bandwidth and regulatory constraints." in *NSDI*, vol. 7, no. 7.2, 2015, pp. 7–8.

[4] K. Kloudas, M. Mamede, N. Preguiça, and R. Rodrigues, "Pixida: optimizing data parallel jobs in wide-area data analytics," *Proceedings of the VLDB Endowment*, vol. 9, no. 2, pp. 72–83, 2015.

[5] Q. Pu, G. Ananthanarayanan, P. Bodik, S. Kandula, A. Akella, P. Bahl, and I. Stoica, "Low latency geo-distributed data analytics," in *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4. ACM, 2015, pp. 421–434.

[6] Z. Hu, B. Li, and J. Luo, "Flutter: Scheduling tasks closer to data across geo-distributed datacenters," in *INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications, IEEE.* IEEE, 2016, pp. 1–9.

[7] L. Chen, S. Liu, B. Li, and B. Li, "Scheduling jobs across geo-distributed datacenters with max-min fairness," *IEEE Transactions on Network Science and Engineering*, 2018.

[8] R. Viswanathan, G. Ananthanarayanan, and A. Akella, "Clarinet: Wan-aware optimization for analytics queries." in *OSDI*, vol. 16, 2016, pp. 435–450.

[9] C.-C. Hung, L. Golubchik, and M. Yu, "Scheduling jobs across geo-distributed datacenters," in *Proceedings of the Sixth ACM Symposium on Cloud Computing.* ACM, 2015, pp. 111–124.

[10] Z. Zhang, L. Cherkasova, and B. T. Loo, "Performance modeling of mapreduce jobs in heterogeneous cloud environments," in *Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on.* IEEE, 2013, pp. 839–846.

[11] K. Wang and M. M. H. Khan, "Performance prediction for apache spark platform," in *High Performance Computing and Communications (HPCC), 2015 IEEE 7th International Symposium on Cyberspace Safety and Security (CSS), 2015 IEEE 12th International Conferen on Embedded Software and Systems*

(ICESS), 2015 IEEE 17th International Conference on. IEEE, 2015, pp. 166–173.

[12] K. Wang, M. M. H. Khan, N. Nguyen, and S. Gokhale, “Modeling interference for apache spark jobs,” in *Cloud Computing (CLOUD), 2016 IEEE 9th International Conference on.* IEEE, 2016, pp. 423–431.

[13] N. Nguyen, M. M. H. Khan, Y. Albayram, and K. Wang, “Understanding the influence of configuration settings: An execution model-driven framework for apache spark platform,” in *Cloud Computing (CLOUD), 2017 IEEE 10th International Conference on.* IEEE, 2017, pp. 802–807.