Towards Optimizing Data Computing Flow in the Cloud

Farshid Farhat, Diman Zad Tootaghaj, Mohammad Arjomand
The Pennsylvania State University
email:{fuf111,dxz149,arjomand} @cse.psu.edu

Abstract

With rapid growth in the amount of unstructured data produced by high performance computing applications, large scale data analytics has recently attracted increasing interest. Processing, managing and analyzing this huge amount of data poses several challenges in cloud and data center computing domain. Especially, conventional frameworks for distributed data analytics are based on the assumption of homogeneity and non-stochastic distribution of different data-processing nodes. The paper argues the fundamental limiting factors for scaling big data computation. It is shown that as the number of series and parallel computing servers increase, the tail (mean and variance) of the job execution time increase. We will first propose a model to predict the response time of highly distributed processing tasks and then propose a new practical computational algorithm to optimize the response time.

1 Introduction

Huge and growing dataset sizes has lead to the emerging field of *Big Data analytics*. As we enter the "petabyte Age", and since data sizes are growing much faster than the processing speeds of computing devices, managing and processing large dataset sizes, scattered over multiple nodes, becomes more challenging [1–5]. In order to process such huge datasets, data computation is usually done in parallel and over multiple nodes. Mapreduce, [6] and its implementations like Hadoop [7] and Dryad [8] are examples of such distributed programming frameworks for distributed processing of very large dataset sizes. These programming frameworks have been optimized for homogeneous hardware architecture, however there exists heterogeneity in the system. Resource sharing which may lead to resource contention, network and memory bandwidth queuing delays, power constraints, inherent heterogeneous workload and computing/data nodes are all different factors which may result in different service rates for different servers in large scale distributed computing. Therefore, we are not able to get the desired performance from the current system.

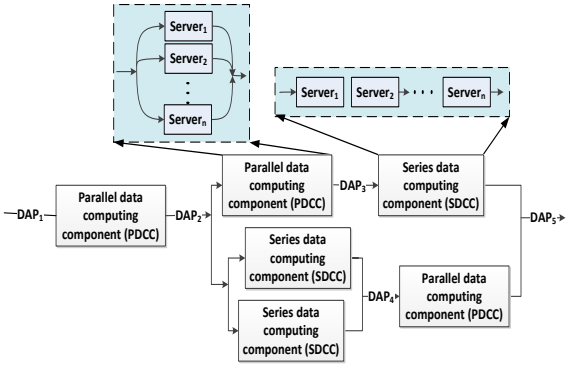


Figure 1: An example of dataflow with both series and parallel components.

Distributed jobs have to confront with performance variation which is caused by resource contention, network contention or heterogeneity in workload or computing nodes. The stochastic performance variation (also known as stragglers) can cause as high as 100x performance degradation on overall job computation time [9, 10]. In general, as the number of parallel/series processing components increases, the mean and variance of total job completion time also increases [9].

While prior works [11–18] have addressed such heterogeneity and stochastic response time of servers in data centers, non-stochastic (deterministic) assumption is still widely used in commercial data centers due to 1) limited mathematical modeling available for such stochastic behavior and 2) lack of a practical method to perform stochastic analysis to optimize job scheduling based on the stochastic system assumption.

Furthermore, each Mapreduce job is a smaller part of the chain for big data analysis. For example, Google

uses a chain of Mapreduce operations for indexing. Figure 1 shows an example dataflow chain which consist of series and parallel components. In this paper's figures, "m" stands for mean and "v" stands for variance. Each fork/join point in Figure 1 is equivalent to a data node that we call it *Data Access Point (DAP)*, which can have a different stochastic distribution for inter-arrival times.

As shown in Figure 1, the incoming data is first accessed by the *Data Computing Component (DCC)* at the data access point, called DAP. Then, there may be a *Serial/Parallel Data Computing Component*, called *SDCC* or *PDCC*, to do the necessary process. Then, the processed data may flow in other branches of the job workflow.

Figures 2a and 2b plot the total end-to-end service time distributions of 10 to 50 server nodes that each have exponential distribution when they run in series. We will explain the figure and our serial computing model in more details in Section 2.1. The first figure shows the cumulative distribution function and the second one shows the probability distribution function. These figures illustrate how the mean and variance of job service time of a series of exponential distribution increase as the number of series components increase. Therefore, the tail of the graph increases as the number of serial or parallel components increase, even if each job has an exponential distribution, the whole system has a long tail distribution which is a limiting factor for scaling big data computation. Mapreduce, is used for batch processing of large-scale datasets. It has been shown that query-based jobs also face the same problem [9, 19].

Problem definition: Consider a system with M heterogeneous servers with different service rates that (collectively) need to process a data workflow (e.g., Figure 1). The objective of resource allocation and task scheduling is to place the servers inside DCCs and distribute the tasks by adjusting the rates of DAPs for better performance. For example, let us assume that we have a 4 servers available at time t with service rates of 2, 4, 6, and 8. Then, the questions we want to answer are which servers should compute which part of the workflow and how much work we can schedule for each of them?

A brief description of the proposed model: We propose to monitor each data access point (DAP), and based on the monitored distribution of each DAP, distribute the jobs such that the total execution time would be minimized. It is shown that any distributed job can be modeled as series and parallel servers [20, 21]. We find an analytical model for service time distribution of the parallel and series servers and frequently monitor the distribution over time. Based on the monitored distribution we find optimum scheduling algorithm to distribute the jobs among servers. To best of our knowledge, this is the first paper that proposes to perform Job scheduling using the stochastic variation of service time of different servers and the dataflow graph.

2 Model Description

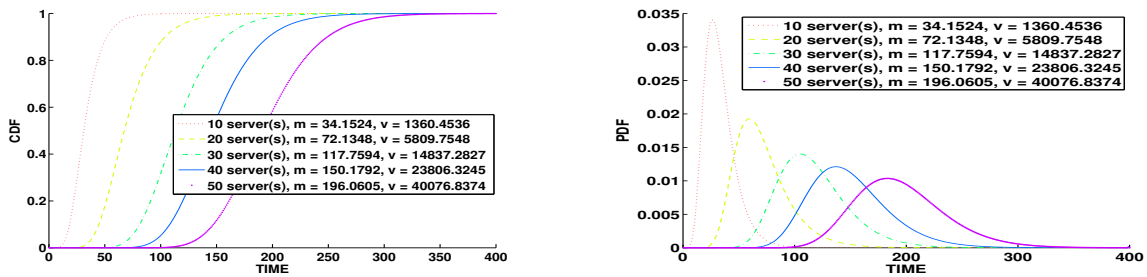
A job is a process that needs to be executed in a specified time. Executing a large dataset in the cloud environment or in a data center may require a sequence of series or parallel data processing phases, where each phase can be run sequentially or in parallel. In fact, in most cases. the workflow of the job is based on its algorithm. Note however that, the amount of resources provided to the job should be allocated and scheduled by the system administrator dynamically.

Previous works have used deterministic (not stochastic) performance metrics [11–13] for each phase and optimized the end-to-end performance of the workflow using computing/processing power of different components in the path. There exists no stochastic modeling of job workflow in the cloud or data center environment. Our proposed model considers, an execution time delay for each server node as well as a waiting time distribution to perform the task. It is assumed that a server is a *queue*, where tasks come for service with a specific service rate.

Most of the servers' waiting time distribution can be modeled as a short/long delayed tail distribution. So the underlying distribution is exponential/pareto or a combination of them (known as multi-modal distribution). The main distributions which have been modeled in this paper are listed in Table 1. Using real data center service time distributions it is shown in [10, 22–27] that the cumulative distribution function of service time of servers can be modeled as six different distributions listed in Table 1. The delayed step function $U(t-T_i)$ specifies the minimum amount of time to complete a task.

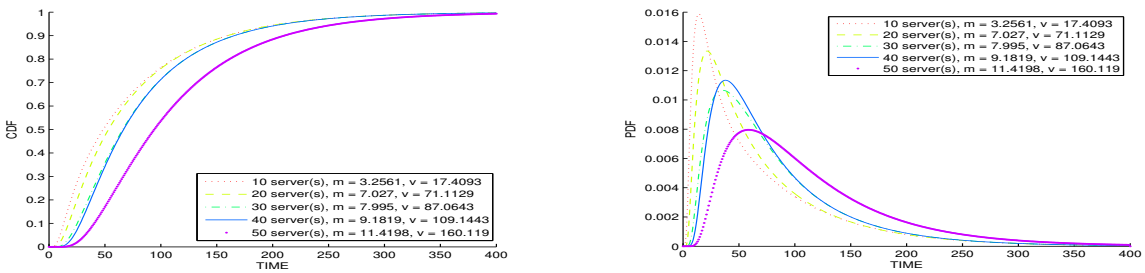
Table 1: Cumulative distribution functions of service time distributions used in the paper.

Distribution name	Cumulative distribution function
Delayed exponential	$F_{de_i}(t) = (1-\alpha e^{-\lambda_i(t-T_i)})U(t-T_i)$
Delayed pareto	$F_{dp_i}(t) = (1-\alpha e^{-\lambda_i(ln(t+1)-T_i)})U(t-T_i)$
Multi-modal delayed exponential (DE)	$F_{mmde}(t) = \sum_{i=1}^{n} p_i F_{de}(t)$; Where $\sum_{i=1}^{n} p_i = 1 \quad \forall i \quad p_i \geq 0$
Multi-modal delayed pareto (DP)	$F_{mmdp}(t) = \sum_{i=1}^{n} p_i F_{dp}(t)$; Where $\sum_{i=1}^{n} p_i = 1 \quad \forall i \quad p_i \geq 0$
Delayed tail	$F_{d_i}(t) = (1-\alpha e^{-\lambda_i(m(t)-T_i)})U(t-T_i)$ Where $m(t)$ is monotonically increasing
Multi-modal delayed tail	$F_{mmd}(t) = \sum_{i=1}^{n} p_i F_{d_i}(t)$; Where $\sum_{i=1}^{n} p_i = 1 \quad \forall i \quad p_i \geq 0$



(a) CDF. (b) PDF.

Figure 2: Job execution time distribution function of 10-50 serial servers.



(a) CDF. (b) PDF.

Figure 3: Job execution time distribution function of 10-50 parallel servers.

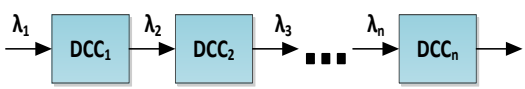


Figure 4: An example of serial DCCs.

2.1 Serial Data Computing Model

In a serial data computing component, the data passes through a set of DCCs sequentially till it gets out from the last one. As shown in Figure 4, the sequential DCCs form a tandem queue with response times of X_1, X_2, ..., X_n. The end-to-end response time of tandem queue of DCCs is $\sum_i X_i$, and consequently the distribution of end-to-end response time would be the convolution of the residual distribution. We have:

$$f_{X_1+X_2+\ldots+X_n}(t) = f_{X_1}(t) * f_{X_2} * \ldots * f_{X_n}(t) = \mathop{\ast}_{n}^{i=1}(f_{X_i}(t)). \tag{1}$$

For example, the sum of two exponential distributions $F_1(t) = 1 - e^{-\lambda_1 t}$ and $F_2(t) = 1 - e^{-\lambda_2 t}$ is

$$F(t) = F_1(t) * F_2(t) = 1 - \frac{\lambda_2}{\lambda_2 - \lambda_1} e^{-\lambda_1 t} + \frac{\lambda_1}{\lambda_2 - \lambda_1} e^{-\lambda_2 t} \tag{2}$$

The effect of job serialization was shown in Figure 2. As the number of serial servers increases, the probability distribution function of end-to-end response time shifts to right, i.e., the mean and variance of the job completion increases.

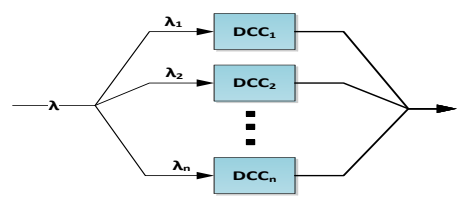


Figure 5: An example of parallel DCCs.

2.2 Parallel Data Computing Model

In a parallel data computing component, the data is partitioned and sent through a set of DCCs in parallel; and the task is completed when the last DCC sends its result to the next joint data access point, as shown in Figure 5. The end-to-end response time (between DAP_1 and DAP_2 in the Figure) of fork-join servers is the $max(X_1, X_2, \ldots, X_n)$, where X_i is the response time of ith DCC. Therefore, we have:

$$\begin{aligned} F_{max(x_1,\ldots,x_n)}(t) &= P(max(X_1,\ldots,X_n) \le t) = \\ &P(X_1 \le t)P(X_2 \le t)\ldots P(X_n \le t) = \\ &F_{X_1}(t)\ldots F_{X_n}(t) = \prod_{i=1}^{n} F_{X_i}(t) \end{aligned} \tag{3}$$

For example, the maximum of two exponential distributions $F_1(t) = 1 - e^{-\lambda_1 t}$ and $F_2(t) = 1 - e^{-\lambda_2 t}$ is

$$F(t) = F_1(t)F_2(t) = (1 - e^{-\lambda_1 t})(1 - e^{-\lambda_2 t}) \tag{4}$$

As shown in Figures 3a and 3b, when the number of parallel servers increases, the mean and variance of job completion time elevates and the PDF moves to the right. However, the magnitude of parallel effect is lower than the serial effect. This result also verifies the effect mentioned in [9].

3 Optimization of Job Workflow

From the administrative point of view, we want to know the end-to-end distribution of the system and we want to optimize the end-to-end response time or maximize throughput.

The goal is to find a resource allocation (placing servers inside DCCs) and task scheduling (adjusting rates of DAPs) for better performance. Here, we aim for throughput or response time; however, our optimization strategy can also be used for other objective functions as well. The main lemma (not proven here because of space limit) is:

Lemma 1. *The global optimization scheme can be divided and conquered; i.e., it can be decomposed into smaller optimization problems on serial and parallel data computation components.*

To have a better throughput, the waiting time of all serial component must be minimum and the same. In other words, we desire to minimize the delay of the SDCC which has the highest delay.

To have a better response time, the parallel DCCs must have the same lowest statistical moment, such as first moment (mean) or second degree moment (variance) or equivalently, we desire to minimize the longest delay of parallel components. The necessary information to manage job workflow is the performance distribution of each server which is gradually updated over the time.

In order to implement data computing flow management, we focus on minimizing response time which is the dual optimization of maximizing the throughput. Finding the best response time of the workflow leads to the problem of finding the best response time of its sequential building blocks. For example, Figure 6 shows a logical view of a job workflow captured by an iterating algorithm in big data analysis. Therefore, the minimization of the response time from DAP_0 to DAP_3 is equivalent to the minimization of the response times of DCC_0, DCC_1 and DCC_2. Now, if DCC_i corresponds to a SDCC (as in DCC_1), we use SDCC optimization algorithm; and if DCC_i corresponds to a PDCC, as in DCC_0 and DCC_2, we employ PDCC optimization algorithm recursively. [1]

As a result of SDCC_allocate algorithm, the faster servers are placed into the DCC with higher data arrival rates. For parallel building blocks we run Algorithm 2 recursively.

[1]Note that there may be other *DCC*s inside a *DCC* as well.

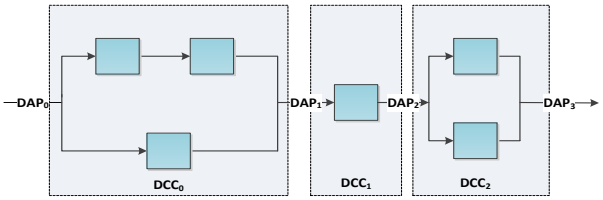


Figure 6: Logical view of a job workflow including serial and parallel DCCs.

Algorithm 1: SDCC_allocate Algorithm.

Input: Available server's response time distribution (RES_Array), and data arrival rates (amount of task) in each DAP (DCCRES_Array).
Output: Server (resource) allocation and data rate (task) scheduling.
1 Sort available servers based on expected values of their response time distribution in descending order in an array (RES_Array)
2 Sort DCCs based on their data arrival rates in ascending order in an array (DCC_Array).

```
/* Allocate the sorted RES_Array to the sorted
   DCC_Array respectively.                      */
for i = 1 → size(DCC_Array) do
    if DCC_Array[i] is a single queue) then
        Place RES_Array[1] in DCC_Array[i]
        Remove the head of RES_Array
    else if DCC_Array[i] == SDCC then
        SDCC_Allocate(RES_Array, DCC_Array[i])
    else if DCC_Array[i] == PDCC then
        PDCC_Allocate(RES_Array, DCC_Array[i])
    end if
end for
```

To perform data computing flow management, we propose Algorithm 3. It is assumed that the logical graph of the job workflow is known using a computational algorithm (out of the scope of this paper). Using Algorithm 3, the DCCs flow is extracted. If there exists a SDCC or

Algorithm 2: PDCC_allocate Algorithm.

Input: Sorted available servers (RES_Array) based on response time distribution in descending order and data arrival rates (amount of task) in each DAP.
Output: Server (resource) allocation and data rate (task) scheduling.

```
if λ_i s are known then
    Sort DCCs list based on their λ_i s as in DCC_Array
    Allocate DCC_Array with RES_Array respectively (Allocate(RES_Array,
    DCC_Array).
end if
if λ_i s are unknown and only their sum (λ) is known then
    Sort DCCs based on the number of internal DAPs in descending order as
    an array (DCC_Array)
    for i = 1 → size(DCC_Array) do
        if (DCC_Array[i] is a single queue) then
            Place RES_Array[1] in DCC_Array[i]
            Remove the head of RES_Array
        else if DCC_Array[i] == SDCC then
            SDCC_Allocate(RES_Array, DCC_Array[i])
        else if DCC_Array[i] == PDCC then
            PDCC_Allocate(RES_Array, DCC_Array[i])
        end if
    end for
end if
```

Rate scheduling: By solving the following equilibrium, we can find the data arrival rates (amount of tasks) sent for each DCC:

$$\lambda = \sum_{i=1}^{n} \lambda_i,$$
$$\lambda_1 RT_{DCC_1} = \lambda_2 RT_{DCC_2} = \ldots = \lambda_n RT_{DCC_n}$$

Where RT is the response time of DCC_i and λ_i is the data arrival rate to DCC_i as shown in Figure 5

Algorithm 3: Management of data computing flows.

Input: Performance distribution of each server and logical graph of the job.
Output: Distribution of workflows.

1. Get the logical workflow of a job based on an iterating algorithm (DCC_Array).
2. Amount of work to be done by a server, i.e. arrival rate of task to each server (RES_Array).
3. Compute power of a server, i.e. recent waiting time distribution of each server.
4. SDDC_allocate(RES_Array, DCC_Array)

PDCC, the algorithm calls Algorithm 1 or 2, respectively, to perform resource allocation/task scheduling.

As shown in Figure 7, we tested our proposed algorithms, the optimal method, and the heuristic baseline algorithm, for the workflow depicted in Figure 6, where DAPs rates were $\lambda_{DAP_0} = 8$, $\lambda_{DAP_1} = 4$, and $\lambda_{DAP_2} = 2$, and available servers have the service rates of 9, 8, 7, 6, 5 and 4. We used different CDFs from Table 1 and the results is valid for all 6 distribution functions. In the heuristic baseline algorithm we use the algorithm that allocates SDCCs sooner than PDCCs and for parallel components, jobs are distributed using the equilibrium equation in Algorithm 2. Heuristic baseline algorithm first allocates better servers to SDCCs (as they become intuitively bottleneck servers), and then allocates PDDCs. Optimal method chooses the best allocation of servers (using exhaustive search over all possible cases) to DCCs and uses optimal task scheduling for PDCCs. To be fair, we used the optimal task scheduling for the heuristic baseline algorithm (however in real systems it may not be true if the scheduler assigns same amount of work to all servers with homogeneous assumption), but our proposed algorithm is as mentioned before.

Table 2 shows the results (mean and variance) of multiple possible scenarios, when the available servers' service time distribution are delayed exponential, delayed pareto, or mix of them. Based on these promising results, we can always get improvement for mean and variance of the system response time over the baseline allocation/scheduling algorithm with a little gap from the optimal choice.

4 Concluding Remarks

In this study, we conducted an analytic study of different dataflows in distributed analysis to monitor data access points and based on service time distribution of servers at each data access point we propose a scheduling algorithm to minimize the total execution time.

While we made our proposed approach simple, more complicated dynamic task scheduling could be used with higher complexity. In fact, the software and hardware concepts in the cloud were modeled or mathematically mapped as queuing theoretic building blocks, while most of such mapping description was skipped as well as some fundamental lemmas.

Knowing the data arrival rates and service rates before execution may have the overhead to run the framework dynamically. This approach forms a bridge between machine learning schemes and analytical schemes. We believe that our proposed approach has lower overhead than the machine learning schemes for managing large number of servers. However, practical issues of the approach have yet to be well investigated.

References

[1] C. Anderson. The petabyte age: because more isnt just more–more is different. *Wired Magazine*, 2008.

[2] V. Mayer-Schönberger and K. Cukier. *Big data: A revolution that will transform how we live, work, and think*. Houghton Mifflin Harcourt, 2013.

[3] M. Arjomand, A. Jadidi, M. T. Kandemir, A. Sivasubramaniam, and C. Das. Mlc pcm main memory with accelerated read. In *International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2016.

[4] M. Arjomand, A. Jadidi, A. Shafiee, and H. Sarbazi-Azad. A morphable phase change memory architecture considering frequent zero values. In *29th International Conference on Computer Design (ICCD)*. IEEE, 2011.

[5] A. Jadidi, M. Arjomand, and H. Sarbazi-Azad. High-endurance and performance-efficient design of hybrid cache architectures through adaptive line replacement. In *Proceedings of the 17th IEEE/ACM international symposium on Low power electronics and design*. IEEE Press, 2011.

[6] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 2008.

[7] `Hadoop. http://lucene.apache.org/hadoop/.`

[8] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *ACM SIGOPS Operating Systems Review*, 2007.

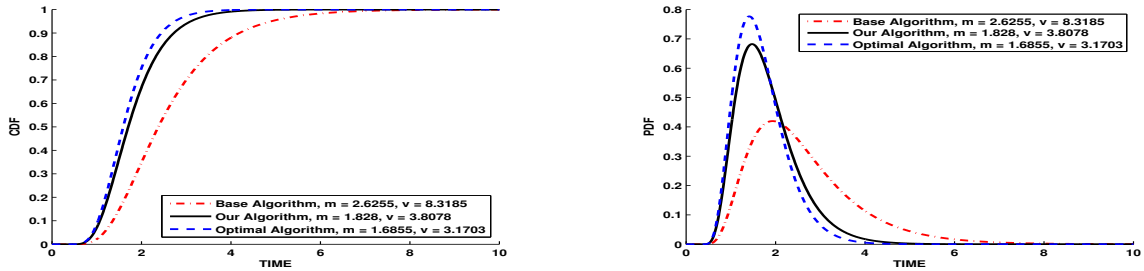
[9] J. Dean and L. A. Barroso. The tail at scale. *Communications of the ACM*, 2013.

[10] G. Ananthanarayanan, S. Kandula, A. G. Greenberg, I. Stoica, Y. Lu, B. Saha, and E. Harris. Reining in the outliers in map-reduce clusters using mantri. In *OSDI*, 2010.

[11] F. Ahmad, S. Chakradhar, A. Raghunathan, and T. N. Vijaykumar. Tarazu: optimizing mapreduce on heterogeneous clusters. In *ACM SIGARCH Computer Architecture News*, 2012.

[12] C. Delimitrou and C. Kozyrakis. Paragon: Qos-aware scheduling for heterogeneous datacenters. *ACM SIGARCH Computer Architecture News*, 2013.

[13] S. Yeo and H-HS Lee. Using mathematical modeling in provisioning a heterogeneous cloud computing environment. *Computer*, 2011.



(a) CDF. (b) PDF.

Figure 7: Comparison of response time distribution between the baseline, optimal and our scheme.

Table 2: Simulation results for different distribution functions listed in Table 1.

Performance metric	**mean**				**Variance**			
	our approcah	optimal	baseline	improvement	our approcah	optimal	baseline	improvement
Scenario 1	1.828	1.6855	2.6255	30.38%	3.8078	3.1703	8.3185	54%
Scenario 2	2.8637	2.5251	5.4125	47.1%	11.4057	8.3924	39.3003	71%
Scenario 3	2.5038	2.09	4.4089	43.2%	8.8881	5.4871	27.741	68%

[14] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica. Improving mapreduce performance in heterogeneous environments. In *OSDI*, 2008.

[15] R. Boutaba, L. Cheng, and Q. Zhang. On cloud computational models and the heterogeneity challenge. *Journal of Internet Services and Applications*, 2012.

[16] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proceedings of the Third ACM Symposium on Cloud Computing*, 2012.

[17] A. Kurve, K. Kotobi, and G. Kesidis. An agent-based framework for performance modeling of an optimistic parallel discrete event simulator. *Complex Adaptive Systems Modeling*, 2013.

[18] K. Kotobi, P. B. Mainwaring, C. S. Tucker, and S. G. Bilén. Data-throughput enhancement using data mining-informed cognitive radio. *Electronics*, 2015.

[19] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica. Why let resources idle? aggressive cloning of jobs with dolly. *Memory*, 2012.

[20] `Google Cloud Platform Blog, Sneak peek: Google Cloud Dataflow, a Cloud-native data processing service. http://googlecloudplatform.blogspot.com.`

[21] J. Zhang, H. Zhou, R. Chen, X. Fan, Z. Guo, H. Lin, J. Y. Li, W. Lin, J. Zhou, and L. Zhou. Optimizing data shuffling in data-parallel computation by understanding user-defined functions. In *NSDI*, 2012.

[22] K. Ren, Y. Kwon, M. Balazinska, and B. Howe. Hadoop's adolescence: an analysis of hadoop usage in scientific workloads. *Proceedings of the VLDB Endowment*, 2013.

[23] F. Farhat, D. Z. Tootaghaj, Y. He, A. Sivasubramaniam, M. T. Kandemir, and C. R. Das. Stochastic modeling and optimization of stragglers. *IEEE transaction on Cloud Computing (TCC)*, 2016.

[24] F. Farhat. Stochastic modeling and optimization of stragglers in mapreduce framework. Master's thesis, The Pennsylvania State University, 2015.

[25] F. Farhat, D. Z. Tootaghaj, A. Sivasubramaniam, M. T. Kandemir, and C. R. Das. Modeling and optimization of straggling mappers. Technical report, Technical Report CSE-14-006, Pennsylvania State University, 2014.

[26] D. Z. Tootaghaj, F. Farhat, M. Arjomand, P. Faraboschi, M. T. Kandemir, A. Sivasubramaniam, and C. R. Das. Evaluating the combined impact of node architecture and cloud workload characteristics on network traffic and performance/cost. In *IEEE international symposium on Workload characterization (IISWC)*, pages 203–212. IEEE, 2015.

[27] D. Z. Tootaghaj. evaluating cloud workload characteristics. Master's thesis, The Pennsylvania State University, 2015.