A Queueing Theory Approach to Pareto Optimal Bags-of-Tasks Scheduling on Clouds

Cosmin Dumitru[1], Ana-Maria Oprescu[1], Miroslav Živković[1], Rob van der Mei[2], Paola Grosso[1], and Cees de Laat[1]

[1] System and Network Engineering Group
University of Amsterdam (UvA)
Amsterdam, The Netherlands
C.Dumitru@uva.nl
[2] Department of Stochastics
Centre for Mathematics and Informatics (CWI)
Amsterdam, The Netherlands
R.D.van.der.Mei@cwi.nl

Abstract. Cloud hosting services offer computing resources which can scale along with the needs of users. When access to data is limited by the network capacity this scalability also becomes limited. To investigate the impact of this limitation we focus on bags–of–tasks where task data is stored outside the cloud and has to be transferred across the network before task execution can commence. The existing bags–of–tasks estimation tools are not able to provide accurate estimates in such a case. We introduce a queuing–network inspired model which successfully models the limited network resources. Based on the Mean–Value Analysis of this model we derive an efficient procedure that results with an estimate of the makespan and the executions costs for a given configuration of cloud virtual machines. We compare the calculated Pareto set with measurements performed in a number of experiments for real–world bags–of–tasks and validate the proposed model and the accuracy of the estimated configurations.

1 Introduction

Bag–of–tasks (BoT) applications are common in science and engineering and are composed of multiple independent tasks, which can be executed without any ordering requirements. Therefore, the execution of a typical BoT application can be parallelized. As the number of tasks within a particular BoT application may be large, the application may also be computationally (i.e. resource) demanding. The execution parallelism and resource demanding properties of BoT applications make them suitable for deployment and execution within the cloud environment. Since the cloud environment has large (theoretically unlimited) resources, the widely–adopted pay–as–you–use model implies the assignment of budgets and/or execution deadlines. Characteristics of tasks, such as the running time, are not given a priori, and therefore need to be estimated [10]. Taking into

account the lack of prior knowledge of the tasks' running times, this presents the challenges for the resource management system with the conflicting goals of minimizing the execution cost while meeting the total execution time (makespan) deadlines.

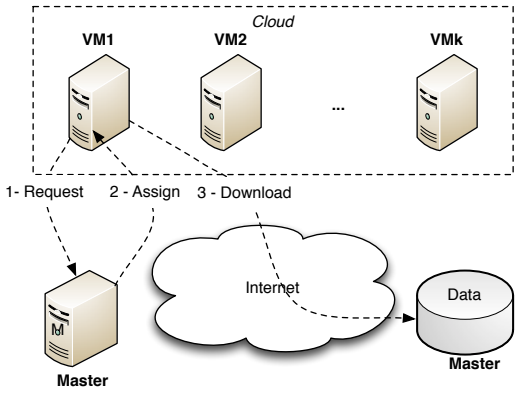

Fig. 1. Execution of Bag–of–Tasks on cloud virtual machines

In general, there are two types of BoT applications, namely *compute–intensive* and *data–intensive* applications. We focus in this paper on *data–intensive* BoT applications where each task requires the *large–sized* data to be available at the location where data processing takes place *before* actual processing. In a typical scenario involving such BoT applications (see Figure 1) the Master (owned by the cloud user) has a BoT, and each task is to be executed by one of the K Virtual Machines (VMs), VM_1, VM_2, ..., VM_K. As the VMs are instantiated in the cloud and become ready, they connect to the Master. When a VM connects (1) , the Master randomly selects a task from a BoT, and assigns (2) it to the VM. In order to accomplish the assigned task, the VM has to retrieve the data of a–priori unknown *large* size via Internet from a remote server (3), and during the retrieval process, this VM may compete for the network and remote server resources with other VMs. Naturally, the more VMs that compete for network and remote server resources, the longer the retrieval time, and consequently, the larger the makespan. Similarly, the larger the data to be retrieved, the longer the retrieval time and the makespan. However, predicting by how much these factors will impact the makespan remains a considerable challenge.

In this paper we analyze the significance of the data transfer performance uncertainty to the makespan. This uncertainty further affects the accuracy of the schedules presented to the user as (nearly) optimal. This is a consequence of the approach in which state–of–the–art schedulers cannot cannot identify the network contention induced by a large number of VMs participating in an execution, or large data transfers (or both). This leads to incomplete executions, or dramatically violated makespan constraints. We derive a queueing–theory based model that allows efficient investigation of the impact of data transfer to the

makespan. Based on the model and performed analysis, we derive the procedure that allows efficient numerical derivation of the makespan, which further allows to calculate the Pareto optimal solutions for execution costs and makespan.

The main contributions of our paper could be summarized as:

- We derive and discuss the queueing–theory based model of the cloud system used for the BoT applications. This model takes into account the data transfer, and requires only the average size of the data set within the BoT. The average size of the data may be estimated using well–known procedure for estimating bags stochastic properties [11].
- We analyze the model using Mean–Value Analysis (MVA) [6] and develop the simplified, yet efficient procedure that allows us to determine the data retrieval time, and to estimate the makespan.
- We validate the proposed model against the traces of two different types of real–world BoT applications executions on real–world clouds. In addition, we use the MVA method to derrive Pareto optimal configurations.

The paper is organized as follows: in Section 2 we describe the related work. In Section 3 we describe the system model which accounts the large data transfers. Further we analyse the proposed model using an MVA approach. Section 4 discusses the results of the model validation, and illustrates the Pareto front of the makespan in case of data–intensive BoT applications using the large data sets. We present our conclusions in Section 5.

2 Related Work

This work is closely related to a number of topics: resource selection and scheduling in clouds,performance prediction, and data-aware scheduling. In this section we provide a short overview of related work.

Efficient resource scheduling with regard to minimizing makespan or other objectives has been explored within the context of cluster, grid and cloud technologies. A common approach assumes full capacity information of available resources and by employing various heuristics optimal schedules are obtained. The majority of approaches just ignore the data access/transfer requirements and expect that the network behaves as an infinite resource.

In [13] the authors consider network resources in the could resource selection phase, but they are performance constant, regardless of the workload. The assumption made here is that input data is replicated across the available resources. A genetic algorithm is used to obtain the Pareto frontier of combination of resources that would lead to optimal schedules for a given workload.

The Budget Aware Task Scheduler,BaTS[11] uses a stochastic approach to determine the workload's properties and uses the collected information to generate an approximated Pareto set of schedules suitable for the workload, along with a predicted makespan [15]. While this system is efficient in predicting the behavior of compute-intensive workloads, the potential impact of the limited network resources on the makespan is ignored.

The scheduler presented in [8] is able to predict the execution time of more complex workloads, like DAG workflows and it is data-aware, but it expects full information on tasks runtime including the data transfer time. Moreover, this data transfer time does not change over time with the addition of new, possibly different resources (scaling up).

When network resources are involved and data access becomes a bottleneck, two popular approaches are taken. One optimizes based on data locality, that is, jobs are scheduled on resources that are close to the data sources [5], [4]. An orthogonal approach replicates data [9], such that the same data is stored at multiple locations and compute jobs which require the same data can be spread across the best available resources, thus lowering the chances of contention. Systems like Gfarm [14] and Hadoop [16] ensure that data is replicated system-wide in order to avoid data access bottlenecks. The replication strategy and the number of replicas influences the performance of the system.

However, both approaches require either compute resources located conveniently close to the data or extra steps (and costs) to replicate the data before the application starts.

None of the approaches described above take into account the changing data transfer time when predicting performance. Besides, to the best of our knowledge, the queue–network models and Mean Value Analysis were not used for the makespan calculation of data–intensive bags–of–tasks.

3 System Model

In this section we introduce our model of the data–intensive BoT system illustrated in Figure 1. First we describe the details of the observed system, then we explain the queueing–theory based model of the system, and we end this section by describing Mean Value Analysis of the given model.

One of the major assumptions for BoT systems is that all tasks from given BoT are independent from one another, i.e. the tasks could be executed without any ordering requirements. The assignment of a single task T_i from a total of N BoT tasks to virtual machines is random, and we neglect the communication overhead (for this assignment) between a particular virtual machine and the master. There are in total K virtual machines, and once the task T_i is assigned to VM_k, $k = 1, 2, \ldots, K$, the virtual machine downloads the data from the data storage. We note the random variable representing the download time of task T_i as T_d, and the expected value of task download time is noted by $T_D = \mathbb{E}[T_d]$.

Once the data corresponding to T_i has been downloaded by VM_k, this virtual machine immediately starts execution of the assigned task. When processing of task T_i is completed, VM_k requests new task assignment from the master. We neglect the time that VM needs to store (i.e. upload) task's output data to a remote destination. As each VM in the system either downloads data or processes the task, the number of tasks (jobs) allowed in the system is constant and equals K. We note the compute rate of VM_k by μ_k, and therefore the average time $\mathbb{E}[S_k]$

a task has been served by VM_k is given as

$$\mathbb{E}[S_k] = \frac{1}{\mu_k} \; k = 1, 2, \ldots, K. \tag{1}$$

Due to the fact that we neglect the upload data process as well as the communication between master and VMs, our system can be modelled as the closed queueing network. It models both the execution process of the virtual machines and the download process from the data storage.

The virtual machines represent a queueing system where every new task arrival experiences immediate service and does not wait – this system is modelled as the one with infinite number of servers, of which at most K are used. The total time of a task at a given VM (i.e. sojourn time) therefore equals to the service time. It is important to notice that virtual machines have different service rates, as typical for a cloud, in which some resources may be faster than the others. The faster services will therefore, *on average*, process more tasks.

As single data storage is used for the data download, the download happens over shared network resources. Therefore it could be modelled as single–server Processor Sharing (PS) queue, in which the server download rate is μ_S. The PS queue that models download process in our case could be either the Discriminatory Processor Sharing (DPS) or Egalitarian Processor Sharing (EPS) queue. This is due to the fact that download rate experienced by a VM_k is limited by the maximum download rate, μ_k^D, and these download rates may be different for different VMs. When the number of download sessions is small, i.e. when the sum of all the service demands at the server is below μ_S, we have DPS. Otherwise, when the number of download sessions is large, the download process is modelled as EPS. In the EPS model, each of download tasks present in the system obtain a fair share of the capacity. In such a case the download rate experienced by VM_k is $\frac{\mu_S}{\#dtasks}$. The data download rate for task T_i experienced by VM_k is given as the following:

$$\begin{array}{ll} \mu_k^D & \text{if} \quad \sum_{l=1}^{\#dtasks} \mu_l^D \leq \mu_S \\ \frac{\mu_S}{\#dtasks} & \text{if} \quad \sum_{l=1}^{\#dtasks} \mu_l^D > \mu_S \end{array} \tag{2}$$

The model we presented can be considered as a closed BCMP queuing network [2], i.e. there are multiple classes of the tasks as their processing rates depend on the class of the task. This is due to the fact that a task is already mapped to a VM of a certain type before it reaches the server. Next to it, the download rates may differ, as given by equation 3. In order to calculate the makespan, we need the expected time, $\mathbb{E}[T]$ a task spends in the system. As the data requests are generated only when the task assigned to VM_k is completed, the expected time $\mathbb{E}[T_k]$ that tasks assigned to VM_k spend in the system, equals to the sum of the expected download time $\mathbb{E}[T_k^D]$, and the expected service time i.e.:

$$\mathbb{E}[T_k] = \mathbb{E}[T_k^D] + \mathbb{E}[S_k] = \mathbb{E}[T_k^D] + \frac{1}{\mu_k} \; k = 1, \ldots, K. \tag{3}$$

The average download times $\mathbb{E}[T_k^D]$ are dependent on the number of download tasks, and using equation 3 we have

$$\mathbb{E}[T_k^D] = \begin{cases} \frac{1}{\mu_k^D} & \text{if} \quad \sum_{l=1}^{\#dtasks} \mu_l^D \leq \mu_S \\ \frac{\mathbb{E}[\#dtasks]}{\mu_S} & \text{if} \quad \sum_{l=1}^{\#dtasks} \mu_l^D > \mu_S \end{cases} \tag{4}$$

In order to evaluate the expected number of download tasks $\mathbb{E}[\#dtasks]$ from equation 3 we would need the equilibrium state probabilities of our system. While methods to obtain a product form for the equilibrium state probabilities exist [3], they require computing all the states of the network and their complexity increases with the number of nodes in the network. The computing of states may take time, which impact the time required for the makespan calculation. Besides, in order to calculate $\mathbb{E}[T_k^D]$ we need information about each task size. In order to solve these two issues we derived an aggregated model, based on the Mean Value Analysis.

3.1 A Mean Value Analysis Approach

The first step in our approach is to transform the given model into the model in which all virtual machines would have the same compute rate ($\bar{\mu}_k = \bar{\mu}$) as well as download rate ($\bar{\mu}_k^D = \bar{\mu}^D$). The second step is to analyse such model for tasks of average size. This is the essence of the Mean Value Analysis (MVA) approach.

Following similar reasoning presented in Chapter 4 of of [10], we still model the virtual machines as the queueing system with the infinite number of servers, of which at most K are used. The aggregated compute rate (μ_{agg}) of this system remains the same, i.e.

$$\mu_{agg} = \sum_{k=1}^{K} w_k \mu_k \quad \text{where} \quad w_k = \frac{\mu_k}{\sum_{k=1}^{K} \mu_k} \tag{5}$$

where w_k represents the probability that some arbitrary task will be executed on machine k in the non–aggregated system. The service rate of a VM_k in this system is therefore

$$\bar{\mu} = \frac{\mu_{agg}}{K}. \tag{6}$$

The similar reasoning holds for the aggregated download rate μ_{agg}^D, as the remains the same, i.e.

$$\mu_{agg}^D = \sum_{k=1}^{K} w_k \mu_k^D \quad \text{where} \quad w_k = \frac{\mu_k^D}{\sum_{k=1}^{K} \mu_k^D}. \tag{7}$$

The maximum download rate of a VM_k in this system is therefore

$$\bar{\mu}^D = \frac{\mu^D_{agg}}{K}. \tag{8}$$

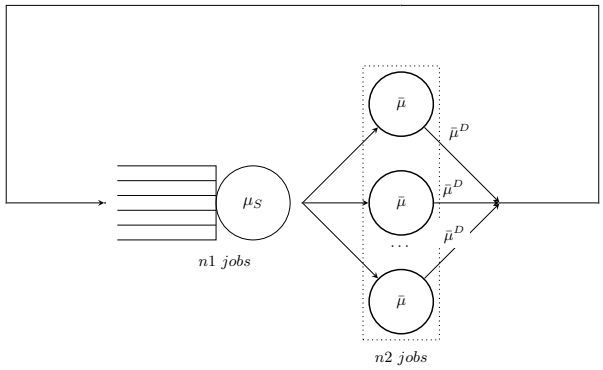

Fig. 2. The aggregated model of the considered system.

As in the original model, the actual data download rate is dependent on the number of tasks that simultaneously access the data storage. The data download rate is now equal for all virtual machines VM_k, and let $\mu_S(j)$ be the service rate of the data storage server when the number of download tasks $\#dtasks = j$. Using equation 8 we obtain the following expression for $\mu_S(j)$ (compare to the equation 3)

$$\mu_S(j) = \begin{cases} \bar{\mu}^D & \text{if} \quad \frac{j}{K} \cdot \mu^D_{agg} \leq \mu_S \\ \frac{\mu_S}{j} & \text{if} \quad \frac{j}{K} \cdot \mu^D_{agg} > \mu_S \end{cases} \tag{9}$$

Due to the aggregation process we can now calculate the stationary probabilities of the system states and we can now easily apply queuing theory numerical methods to derive the parameters of the system. The state of the system is described as (n_1, n_2) where n_1 represents the number of the tasks that are downloaded while n_2 represents the number of the tasks that are processed by (n_2) virtual machines. It holds that $n_1 + n_2 = K$, and $n_1, n_2 > 0$. Let $\pi_1(j|K)$ be the conditional probability that the number of downloading tasks is j under condition that the total number of tasks in the network is K. We define $\pi_2(j|K)$ accordingly.

We are mostly interested in the mean service time experienced by an arriving job at the data storage node, i.e. the average download time. For this we will use the Mean Value Analysis for single chain product form closed networks [6]. The MVA analysis is based on two important results from the queuing theory: *the arrival theorem* [12, 6] and *Little's Law* [7]. By using these fundamental theorems, a series of recurrent relations can be derived for the mean service times, arrival rates and queue lengths for all nodes in the network.

From the arrival theorem we obtain the expected download rate when there are K tasks in the network as the following

$$\mathbb{E}[T^D(K)] = \sum_{j=1}^{K} \pi_1(j-1|K-1)\frac{j}{\mu_S(j)} \tag{10}$$

As all VMs have the same compute rate, the expected service time is constant, i.e.

$$\mathbb{E}[S] = \frac{1}{\mu_{agg}}. \tag{11}$$

The *visit rate* is defined as the mean number of visits made by a task at download server (v_D) or aggregated virtual machines (v_S). In our case, $v_D = v_S = \frac{1}{2}$ as the number of arrivals at download server and aggregated virtual machines is the same. From Little's Law we obtain the total system arrival rate, i.e. *throughput* of the system with K jobs:

$$\lambda(K) = \frac{K}{v_D\mathbb{E}[T^D(K)] + v_S\mathbb{E}[S]} = \frac{K}{\frac{1}{2}\mathbb{E}[T^D(K)] + \frac{1}{2}\mathbb{E}[S]}. \tag{12}$$

The queue length distribution at the download server, i.e. PS queue can be determined from

$$\pi_1(j|K) = \frac{v_1 \cdot \lambda(K)}{\mu_S(j)}\pi_1(j-1|K-1), j = 1, \ldots K. \tag{13}$$

The probability of an empty PS queue is derived from:

$$\pi_1(0|K) = 1 - \sum_{j=1}^{K} \pi_1(j|K). \tag{14}$$

Using recursive formulae 10–14 we can derive $\mathbb{E}[T^D(K)]$. For the total of N tasks within the BoT, the total makespan obtained using the MVA is

$$M = \frac{N}{\frac{K}{\mathbb{E}[T^D(K)]+\mathbb{E}[S]}}. \tag{15}$$

The computation complexity of the MVA-based estimation algorithm is $\mathrm{O}\left(K^2\right)$ where K is the total number of machines. In practice this number is small. This trait makes the MVA approach well suited for estimating the Pareto front of optimal configurations for a given workload.

4 Evaluation and Discussion

In this section we present the empirical evaluation of the MVA makespan prediction method introduced in the previous section. All experiments were performed using the Amazon EC2 [1] cloud region Europe. The storage server hosting the input data was located in the Netherlands. For instance reservation and task execution we used the Budget-aware Task Scheduler[11]. The characteristics of the Amazon EC2 instance types used in our experiments are presented in Table 1.

Table 1. Amazon EC2 Instance Details

Type	ECU	Memory(GB)	Network	Cost($/h)
m1.s	1	1.7	Low	0.65
m1.m	2	3.75	Moderate	0.130
c1.m	5	1.7	Moderate	0.165

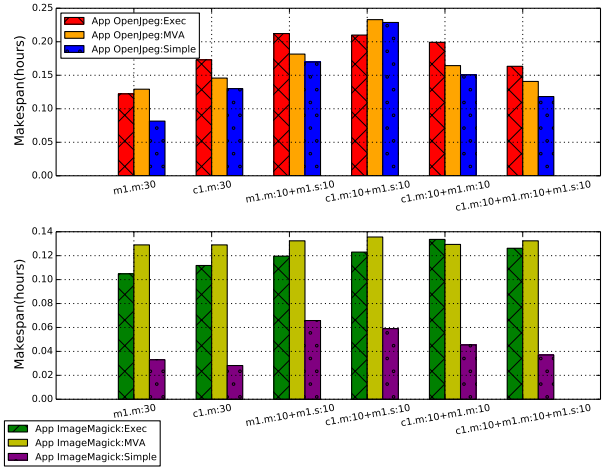

Fig. 3. Measured (exec), MVA Predicted and simple predicted makespans for large schedules

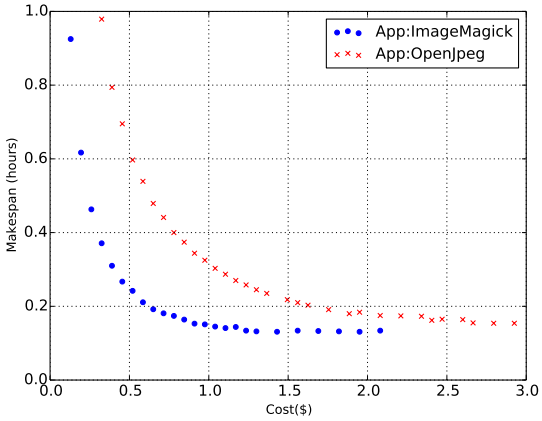

Fig. 4. Pareto fronts for two application types: OpenJpeg and ImageMagick

Applications We considered two image processing applications: 1) *OpenJpeg*, a JPEG2000 software encoder and 2) an application part of the ImageMagick suite, which compresses images to the JPEG format. The input data consisted of the first 1500 TIFF image frames in 4K resolution of the open source movie *Sintel*.

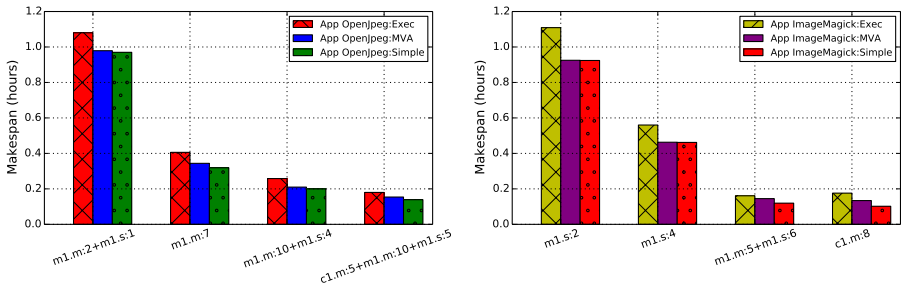

Fig. 5. Measured (exec), MVA Predicted and simple predicted makespans for schedules selected from the Pareto fronts

The average file size was 23.3MB. For both applications, we sampled the expected performance of each instance type considered in Table 1 using BaTS' sampling module. During the sampling procedure, we also performed network bandwidth measurements to obtain the data storage server's capacity. According to our sampling results, while the input data is the same, the *OpenJpeg* application has a higher average execution time since the compression algorithm used is more computationally-intensive.

Figure 3 presents the makespan estimation of three methods applied to large schedules, i.e. each has 30 virtual machines. In the execution method, the schedule is executed and the makespan is recorded .The MVA method follows the steps described in the the previous section. The simple makespan estimation uses only the values observed at sampling and computes the makespan as if the network would not reach contention regardless of the number of virtual machines. This method provides reasonable accuracy if the data storage server does not encounter contention at any point during the execution. For the *ImageMagick* application the simple method proves to be inaccurate, while MVA comes very close to the real makespan. The MVA method is able to include the fact that data storage server has become the bottle neck. Due to the demanding nature of *OpenJpeg* only the m1.m:30 schedule (the fastest schedule) is able to saturate the server and the expected performance is below the one estimated by the simple method. In this case MVA predicts a makespan close to the observed value. We now apply the MVA method to determine the Pareto front.

Figure 4 shows the Pareto fronts for the two applications. The Pareto fronts were obtained by exhaustively estimating the makespan and budget of all possible schedules, within the limits of maximum 10 instances per type, next to computing the non-dominated set of schedules. As the maximum number of instances and instance types increases, this approach becomes extremely slow (the total number of schedules grows exponentially). Pareto frontier (PF) approximations algorithms exist [15], but their use is beyond the scope of this paper. Each point on the graph represents an unique schedule with a corresponding cost and makespan. We observe that each PF exhibits a 'tipping point', that is a point in the objective space from which the speedup obtained by selecting a more expensive schedule decreases considerably: the $1 mark for *ImageMag-*

ick and $1.5 for *OpenJpeg*), respectively. For each application, the respective PF shows a clustering tendency towards the schedule in which the data storage server utilization reaches 100%.

To empirically evaluate the quality of the obtained Pareto fronts, we selected from each Pareto front four schedules: the cheapest schedule, the cheapest from the group of very fast schedules, that is the ones at the right of the 'tipping point', and two other schedules such that they equally divide the price interval between first two selected schedules. Next, we executed the schedules selected from the Pareto fronts. Figure 5 shows the observed makespan after executing the schedule (exec), the simple makespan estimation and the MVA makespan estimation for both types of applications considered. The column labels represent the schedule, type and number of instances.

We then executed the selected schedules. Figure 5 shows the observed makespan after executing the schedule (exec), the simple makespan estimation and the MVA makespan estimation for both types of applications and considered. The column labels represent the schedule, type and number of instances.

For all the schedules the simple and MVA estimates are very close to each other. This is due to the special properties held by the schedules located on the Pareto front. Some of the schedules on the front are the cheapest, and they are never able to saturate the server(`m1.s:2, m1.s:4`), thus the MVA method is equivalent to the simple estimation of the makespan. On the other hand the Pareto front also contains schedules which saturate the server but do not overload it. In other words , the utilization of the server reaches 100% for these schedules but virtual machines rarely compete for the data storage server's resources. If the virtual machines would start to compete, the Pareto optimal property of the schedule would be jeopardized as extra cost (when compared to another schedule) would yield too little speedup. The MVA procedure requires information about the mean behavior of the system's components and thus no other statistical properties can be derived, besides means. While this can be seen as a limitation of the prediction ability of our model, it makes it on the other hand very robust and computationally efficient. As future work we plan to model the system as a more complex queueing network, which would allow us to obtain other properties like for instance service time distributions, etc.

5 Conclusions and Future work

We have presented the theoretical model of a system which executes data-intensive bags–of–tasks in a cloud computing environment. The empirical evaluation of the model shows promising results with regard to makespan estimation for various combinations and numbers of cloud instances in the presence of limited network resources. We have shown how this robust method can be applied to an existing scheduler to obtain Pareto fronts for data intensive bags–of–tasks workloads.

References

1. Amazon ec2 - amazon elastic compute cloud. `https://aws.amazon.com/ec2/`. Accessed: 2014-01-27.
2. F. Baskett, K. M. Chandy, R. R. Muntz, and F. G. Palacios. Open, closed, and mixed networks of queues with different classes of customers. *J. ACM*, 22(2):248–260, apr 1975.
3. G. Bolch, S. Greiner, H. de Meer, and K. S. Trivedi. *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*. Wiley-Interscience, New York, NY, USA, 1998.
4. W. Cirne, D. Paranhos, L. Costa, E. Santos-Neto, F. Brasileiro, J. Sauve, F. A. B. Silva, C. Barros, and C. Silveira. Running bag-of-tasks applications on computational grids: the mygrid approach. In *Parallel Processing, 2003. Proceedings. 2003 International Conference on*, pages 407–416, 2003.
5. J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke. Condor-g: A computation management agent for multi-institutional grids. *Cluster Computing*, 5(3):237–246, jul 2002.
6. S. S. Lavenberg. *Computer Performance Modeling Handbook*. Academic Press, Inc., Orlando, FL, USA, 1983.
7. J. D. C. Little. A proof for the queuing formula: L = λ w. *Operations Research*, 9(3):383–387, 1961.
8. M. Mao and M. Humphrey. Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 49:1–49:12, New York, NY, USA, 2011. ACM.
9. R. McClatchey, A. Anjum, H. Stockinger, A. Ali, I. Willers, and M. Thomas. Data intensive and network aware (diana) grid scheduling. *Journal of Grid Computing*, 5(1):43–64, 2007.
10. A.-M. Oprescu. *Stochastic Approaches to Self-Adaptive Application Execution on Clouds*. PhD thesis, Amsterdam: Vrije Universiteit, 2013.
11. A.-M. Oprescu, T. Kielmann, and H. Leahu. Budget estimation and control for bag-of-tasks scheduling in clouds. *Parallel Processing Letters*, 21(02):219–243, 2011.
12. M. Reiser and S. S. Lavenberg. Mean-value analysis of closed multichain queuing networks. *J. ACM*, 27(2):313–322, apr 1980.
13. J. Taheri, A. Y. Zomaya, H. J. Siegel, and Z. Tari. Pareto frontier for job execution and data transfer time in hybrid clouds. *Future Generation Computer Systems*, (0):–, 2013.
14. A. Takefusa, O. Tatebe, S. Matsuoka, and Y. Morita. Performance analysis of scheduling and replication algorithms on grid datafarm architecture for high-energy physics applications. In *HPDC*, volume 3, page 34, 2003.
15. A. Vintila, A.-M. Oprescu, and T. Kielmann. Fast (re-)configuration of mixed on-demand and spot instance pools for high-throughput computing. In *Proceedings of the First ACM Workshop on Optimization Techniques for Resources Management in Clouds*, ORMaCloud '13, pages 25–32, New York, NY, USA, 2013. ACM.
16. T. White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 1st edition, 2009.