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 in 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 [12]. Taking into

F. Silva et al. (Eds.): Euro-Par 2014, LNCS 8632, pp. 162–173, 2014.

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 deadlines. 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, 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 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.

- We derive and discuss a 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 [12].
- We analyze the model using Mean–Value Analysis (MVA) [8] 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 derive 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 for 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 [14] the authors consider network resources in the cloud 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) [12] 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 [16]. 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 [10] 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 [7], [6]. An orthogonal approach replicates data [11], 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 [15] and Hadoop [17] 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 previously described. 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 $\mathbb{E}[S_k] = \dfrac{1}{\mu_k}$, $k = 1, 2, \ldots, K$.

Due to the fact that we neglect the upload data process as well as the communication between master and VMs, our system can be modeled as the closed queueing network. The VMs represent a queueing system where every new task arrival experiences immediate service and does not wait – this system is modeled as the one with infinite number of servers, of which at most K are used.

As single data storage is used for the data download, the download happens over shared network resources. Therefore it could be modeled 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 modeled as EPS. In the EPS model, each of the 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:

$$\mu_k^D \quad \text{if} \quad \sum_{l=1}^{\#dtasks} \mu_l^D \leq \mu_S \quad \text{and} \quad \frac{\mu_S}{\#dtasks} \quad \text{if} \quad \sum_{l=1}^{\#dtasks} \mu_l^D > \mu_S \qquad (1)$$

The model we presented can be considered as a closed BCMP queuing network [4], 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 1. 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.:

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

The average download times $\mathbb{E}[T_k^D]$ are dependent on the number of download tasks, and using equation 1 we have

$$\mathbb{E}[T_k^D] = \begin{cases} \frac{1}{\mu_k^D} & \text{if} \quad \sum_{l=1}^{\#dtasks} \mu_l^D \leq \mu_S \\ \frac{\mathbb{E}[\#dtasks]}{\mu_S} & \text{if} \quad \sum_{l=1}^{\#dtasks} \mu_l^D > \mu_S \end{cases} \tag{3}$$

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 [5], 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 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.

We model the VMs 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,

$$\mu_{agg} = \sum_{k=1}^{K} w_k \mu_k \quad \text{where} \quad w_k = \frac{\mu_k}{\sum_{k=1}^{K} \mu_k} \tag{4}$$

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 VM_k is

$$\bar{\mu} = \frac{\mu_{agg}}{K}. \tag{5}$$

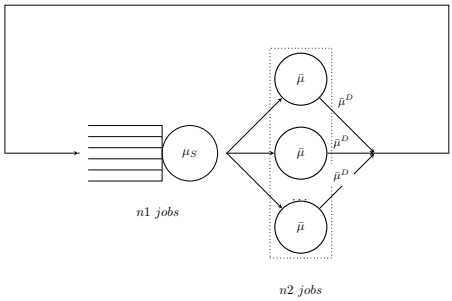

Fig. 1. The aggregated model of the considered system

The similar reasoning holds for the aggregated download rate μ^D_{agg}, i.e.

$$\mu^D_{agg} = \sum_{k=1}^{K} w_k \mu^D_k \quad \text{where} \quad w_k = \frac{\mu^D_k}{\sum_{k=1}^{K} \mu^D_k}. \tag{6}$$

The maximum download rate of VM_k in this system is therefore

$$\bar{\mu}^D = \frac{\mu^D_{agg}}{K}. \tag{7}$$

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 7 we obtain the following expression for $\mu_S(j)$

$$\mu_S(j) = \begin{cases} \bar{\mu}^D & \text{if} \quad \frac{j}{K} \cdot \mu^D_{agg} \leq \mu_S \\ \frac{\mu_S}{j} & \text{if} \quad \frac{j}{K} \cdot \mu^D_{agg} > \mu_S \end{cases} \tag{8}$$

Due to the aggregation process we can now calculate the stationary probabilities of the system states. The system state 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) VMs. It holds that $n_1 + n_2 = K$, and $n_1, n_2 \geq 0$. Let $\pi_1(j|K)$ be the conditional probability that the number of download tasks is j under condition that the total number of tasks in the network is K. We define $\pi_2(j|K)$ accordingly. The mean service time experienced by an arriving job at the data storage node (the average download time) is derived using MVA for the single chain product form closed networks [8]. The MVA analysis is based on two important results from the queuing theory: *the arrival theorem* [13,8] and *Little's Law* [9].

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

$$\mathbb{E}[T^D(K)] = \sum_{j=1}^{K} \pi_1(j-1|K-1) \frac{j}{\mu_S(j)} \tag{9}$$

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

$$\mathbb{E}[S] = \frac{1}{\mu_{agg}}. \tag{10}$$

The *visit rate* is defined as the mean number of visits made by a task at the 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 the download server and the aggregated virtual machines are the same. From Little's Law we obtain the total system arrival rate, i.e. *throughput* of the system with K jobs:

$$\lambda(K) = \frac{K}{v_D\mathbb{E}[T^D(K)] + v_S\mathbb{E}[S]} = \frac{K}{\frac{1}{2}\mathbb{E}[T^D(K)] + \frac{1}{2}\mathbb{E}[S]}. \tag{11}$$

The queue length distribution at the download server is derived from

$$\pi_1(j|K) = \frac{v_1 \cdot \lambda(K)}{\mu_S(j)}\pi_1(j-1|K-1), j = 1, \ldots K. \tag{12}$$

The probability of an empty queue is derived from

$$\pi_1(0|K) = 1 - \sum_{j=1}^{K} \pi_1(j|K). \tag{13}$$

Using recurrence formulae 9–13 we can derive $\mathbb{E}[T^D(K)]$. For a total of N tasks within the BoT, the total makespan obtained using the MVA is

$$M = \frac{N}{\frac{K}{\mathbb{E}[T^D(K)]+\mathbb{E}[S]}}. \tag{14}$$

The computation complexity of the MVA-based estimation algorithm is $\mathrm{O}(K^2)$ where K is the number of VMs. As in practice K is relatively small, the MVA approach is well–suited to estimate the Pareto frontier of optimal configurations for a given workload.

4 Evaluation and Discussion

We evaluate the accuracy of our MVA-based prediction procedure for data-intensive bags–of–tasks using an experimental setup consisting of two real-world applications and multiple cloud instance types. We also investigate the efficiency of our MVA-based procedure when employed towards constructing Pareto fronts.

All experiments were performed using the Amazon EC2 [1] cloud region Ireland. The characteristics of the Amazon EC2 instance types used in our experiments are presented in Table 1. The compute performance of each instance consists of the number of virtual CPUs (vcpus) and their allocated shares, ECU (EC2 Compute Unit), the equivalent of a 2007 AMD Opteron CPU. We chose to focus on these three types because they exhibit different computation-to-network-to-price ratios and therefore allow us to analyze the behavior of the MVA-based procedure in different real-world scenarios. The storage server hosting the input data was located in the Netherlands. For instance reservation and task execution we used the Budget- and Time-constrained Scheduler[12].

Table 1. Amazon EC2 Instance Details

Type	CPUs (ECU)	Memory(GB)	Network	Cost($/h)
m1.s	1(1)	1.7	Low	0.047
m1.m	1(2)	3.75	Moderate	0.095
m1.l	2(4)	7.5	Moderate	0.190

Applications. We considered two image processing applications, each displaying a different compute-to-data ratio: *OpenJpeg* a JPEG2000 software encoder [3] and a *ImageMagick* suite component, which compresses images to the JPEG format [2] and applies a sharpening filter. The input data used for our experiments consisted in a subset of 7500 TIFF image frames in 4K resolution of the open source movie *Sintel*. The average file size was 24.3 MB. For both applications, we estimated the expected performance of each EC2 instance type (see Table 1) using BaTS' sampling module. During the sampling procedure, we also performed network bandwidth measurements to assess the storage server's capacity. We remark that, according to our sampling results, for the same input data, the *OpenJpeg* application has a higher average execution time since the compression algorithm used is more computationally-intensive.

Experiments. To evaluate the accuracy of our MVA-based prediction procedure, we compare it against the data-oblivious prediction mechanism of BaTS, referred to as 'simple', and against real executions ("exec") of several scenarios having the same input data (bag), but different cloud instance configurations:

1-1-1 consists in one instance of each type: m1.s, m1.m, m1.l
5-5-5 consists in five instances of each type: m1.s, m1.m, m1.l
10-10-10 consists in ten instances of each type: m1.s, m1.m, m1.l

Since real executions are subject to external noise (such as network traffic or cloud instance performance variability), we repeat the execution of each scenario three times and derive corresponding error bars to obtain the 'exec' makespans.

All results for both types of applications are collected in Figure 2. For each configuration, we present the MVA-based makespan estimate, the 'simple' makespan estimate and the 'exec' makespan (with error bars). Each configuration is labeled using the types and respective number of instances, in the following format: `type:no_instances[+type:no_instances[...]]`. All executions were performed three times and the makespans averaged. The variance of each execution was relatively low(0.10-0.20), especially for the 'larger' configurations. In the case of 'small' (3 machines) configurations the variance is slightly higher(0.20-0.25). We assume that this is due to both varying network conditions and to the slight variability in performance of the instances. The cloud provider is not able to provide a perfectly identical instance in terms of

performance due to the shared environment. Also small configurations are more sensitive to varying Internet conditions.

We selected these three configurations as they offer a good insight with regard to the behavior of the MVA prediction method in the presence of varying numbers and types of instances. The '1-1-1' configuration has a low number of instances and thus can be used to benchmark the behavior of both the MVA and simple prediction methods. The '5-5-5' configuration starts to encounter contention at the storage server, especially in the case of the *ImageMagick* application, which as previously mentioned, exhibits a lower compute-to-data ratio. We already see here that the 'simple' estimation is no longer accurate enough. The '10-10-10' configuration manages to saturate the storage server in the case of both applications. The MVA method is able to include the fact that the data storage server has become the bottleneck. In all cases the MVA value is close to the measured execution time. This shows that the simplification we have made in our model, where all the different types of instances are aggregated and then homogenized does not considerably affect the accuracy of the MVA method.

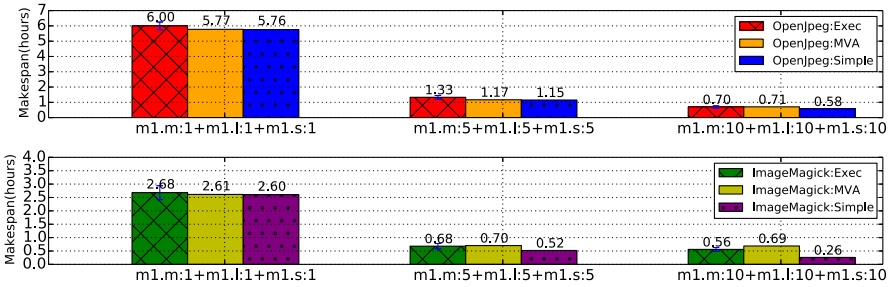

Fig. 2. Measured (exec), MVA Predicted and simple predicted makespans for three configurations

We can now use this result to apply the MVA method to a real scenario in which the user is faced with the task of selecting from a list of configurations, which exhibit different performance and cost. We obtained the Pareto fronts (PFs) of each application, using both the MVA-based and 'simple' estimates, as shown in Figure 3. Each point in the graph represents an unique configuration with its corresponding cost and makespan. The PFs were obtained by exhaustively computing the makespan and budget estimates of all possible configurations, considering a maximum of 10 instances per type, and then selecting the non-dominated set of configuration, i.e. for a configuration from the Pareto Set is . As the maximum number of instances and instance types increases, this approach becomes extremely slow (the total number of configurations grows exponentially). However, here we focus on the efficiency of employing our MVA-based method when constructing PFs and further usage of approximations algorithms is beyond the scope of this paper. In the case of the PF of the *ImageMagick* application we observe a 'tipping point', i.e. a point in the objective space where the

speedup obtained by selecting a more expensive configurations starts decreasing considerably. This is less visible in the case of the *OpenJpeg* application, as the saturation point is not fully achieved not even in the case of the most expensive configuration. This is related, as previously mentioned, to the different compute-to-data ratio of the application.

The 'simple' PF offers a set of configurations which, as empirically shown in the first set of experiments does not represent the 'true' Pareto Front, due to the naive method's inaccuracy in the presence of network bottlenecks. By selecting a configuration from this set, the user could potentially make inefficient use of his budget.

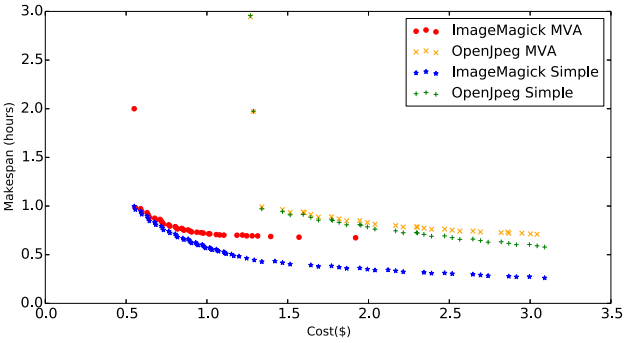

Fig. 3. Pareto fronts for two application types: OpenJpeg and ImageMagick

To empirically evaluate the accuracy of each MVA-based Pareto front, we selected for real execution four configurations: the global cheapest, the cheapest from the group of very fast schedules, i.e. the ones at the right of the 'tipping point', and two other configurations such that they equally divide the price interval between the first two selected configurations. Figure 4 shows the execution makespan (exec), the 'simple' makespan estimate and the MVA-based makespan estimate for each configurations and for each application considered. Each configuration was executed three times and Figure 4 presents the average over the three executions together with the error. Again, we remark that the variance is small, similar to that observed for the first presented experiments.

For all the configurations, the execution times, and both the MVA and simple estimates are close to each other. This is due to the special properties held by the schedules located on the Pareto front. These configurations make best use of the available resources and inherently avoid contention; when contention is reached, the configuration is less efficient with respect to cost and makespan and therefore would not be present in the non-dominated set of configurations (Pareto front).

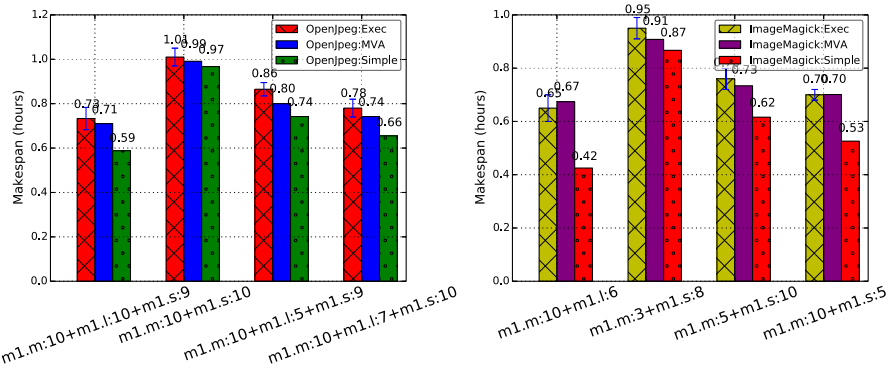

Fig. 4. Measured (exec), MVA-based and 'simple' predicted makespans for Pareto front selected configurations

5 Conclusions and Future work

In this paper we have presented the theoretical model of a system which executes data–intensive bags–of–tasks in a cloud computing environment with data access bottlenecks. The empirical evaluation of the model shows promising results with respect to makespan estimation for various combinations of cloud instances in the presence of limited network resources. We showed how this method (MVA) can be successfully applied to an existing scheduler to obtain Pareto fronts for data–intensive bags–of–tasks workloads. 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 more properties of the system, such as service time distributions.

Funding has been provided by the Dutch national research program COMMIT.

References

1. Amazon ec2 - amazon elastic compute cloud, `https://aws.amazon.com/ec2/` (accessed: January 27, 2014)
2. Imagemagick: Convert, edit, or compose bitmap images, `http://www.imagemagick.org/` (accessed: Januray 27, 2014)
3. Openjpeg - jpeg2000 codec, `http://www.openjpeg.org/` (accessed: January 27, 2014)
4. Baskett, F., Chandy, K.M., Muntz, R.R., Palacios, F.G.: Open, closed, and mixed networks of queues with different classes of customers. J. ACM 22(2), 248–260 (1975)

5. Bolch, G., Greiner, S., de Meer, H., Trivedi, K.S.: Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications. Wiley-Interscience, New York (1998)
6. Cirne, W., Paranhos, D., Costa, L., Santos-Neto, E., Brasileiro, F., Sauve, J., Silva, F.A.B., Barros, C., Silveira, C.: Running bag-of-tasks applications on computational grids: The mygrid approach. In: Proceedings of the 2003 International Conference on Parallel Processing, 2003, pp. 407–416 (2003)
7. Frey, J., Tannenbaum, T., Livny, M., Foster, I., Tuecke, S.: Condor-g: A computation management agent for multi-institutional grids. Cluster Computing 5(3), 237–246 (2002)
8. Lavenberg, S.S.: Computer Performance Modeling Handbook. Academic Press, Inc., Orlando (1983)
9. Little, J.D.C.: A proof for the queuing formula: L = λ w. Operations Research 9(3), 383–387 (1961)
10. Mao, M., Humphrey, M.: 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 2011, p. 49:1–49:12. ACM, New York (2011)
11. McClatchey, R., Anjum, A., Stockinger, H., Ali, A., Willers, I., Thomas, M.: Data intensive and network aware (diana) grid scheduling. Journal of Grid Computing 5(1), 43–64 (2007)
12. Oprescu, A.-M., Kielmann, T., Leahu, H.: Budget estimation and control for bag-of-tasks scheduling in clouds. Parallel Processing Letters 21(02), 219–243 (2011)
13. Reiser, M., Lavenberg, S.S.: Mean-value analysis of closed multichain queuing networks. J. ACM 27(2), 313–322 (1980)
14. Taheri, J., Zomaya, A.Y., Siegel, H.J., Tari, Z.: Pareto frontier for job execution and data transfer time in hybrid clouds. Future Generation Computer Systems (2013)
15. Takefusa, A., Tatebe, O., Matsuoka, S., Morita, Y.: Performance analysis of scheduling and replication algorithms on grid datafarm architecture for high-energy physics applications. In: HPDC, vol. 3, p. 34 (2003)
16. Vintila, A., Oprescu, A.-M., Kielmann, T.: 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 2013, pp. 25–32. ACM, New York (2013)
17. White, T.: Hadoop: The Definitive Guide, 1st edn. O'Reilly Media, Inc. (2009)