Cost-Efficient High-Performance Internet-Scale Data Analytics over Multi-Cloud Environments

Shigeru Imai, Stacy Patterson, and Carlos A. Varela
Department of Computer Science
Rensselaer Polytechnic Institute
$\{$*imais,sep,cvarela*$\}$*@cs.rpi.edu*

***Abstract*—To analyze data distributed across the world, one can use distributed computing power to take advantage of data locality and achieve higher throughput. The multi-cloud model, a composition of multiple clouds, can provide cost-effective computing resources to process such distributed data. As multi-cloud becomes more and more accessible from cloud users, the use of MapReduce/Hadoop over multi-cloud is emerging; however, existing work has two issues in principle. First, it mainly focuses on maximizing throughput by improving data locality, but the perspective of cost optimization is missing. Second, conventional centralized optimization methods would not be able to scale well in multi-cloud environments due to its highly dynamic nature. We plan to solve the first issue by formalizing an optimization framework for MapReduce over multi-cloud including virtual machine and data transfer costs, and then the second issue by creating decentralized resource management middleware that considers multi-criteria (cost and performance) optimization. This paper reports progress we have made so far on these two directions.**

***Keywords*-multi-cloud; optimization; data analytics;**

I. Introduction

As demand for large-scale data processing grows in many application areas, data is logged, collected, and processed globally. For example, seismographic data is collected by sensors at various locations; financial data is generated from stock markets all around the world; and an enormous number of web sites are created and hosted virtually everywhere in the world. These data sets can be very big depending on the number of data sources and the data production rate. Or, often times, they are big in nature such as crowd-sourced databases, for example, Wikipedia. To process these globally distributed "Big Data" in a timely manner, one can take advantage of data locality by using distributed public cloud data centers. Not only public clouds, but also private clouds can be part of distributed computing resources. By connecting these multiple clouds, we can construct a multi-cloud environment [1] that works as a foundation of Internet-scale distributed data analytics. Unlike the federate cloud model requires an agreement between multiple cloud providers to give users transparent access to the cloud, the multi-cloud model is more client-centric. That is, in the multi-cloud model, interoperability between multiple clouds is maintained by the user using libraries or third party brokers [2].

MapReduce's [3] open-source implementation Hadoop [4] has been successfully used in data analytics to date, due to its simple programming abstraction as well as scalable and fault-tolerant architecture. As multi-cloud environments become more and more accessible, deploying MapReduce applications over multi-cloud is emerging [5], [6], [7], [8]. We have identified two main issues for MapReduce-based data analytics over multi-cloud. First, existing work on multi-cloud mainly focuses on maximizing throughput by improving data locality [5], [8], but the perspective of cost optimization is missing. To consider the cost and running time of applications doing the optimization process, we need to focus not only on data locality, but also on data transfer costs defined by multiple different cloud providers' cost models. Second, existing resource provisioning optimization methods for single clouds [9], [10] require centralized knowledge. Namely, a single resource scheduler collects required information for optimization from distributed cloud resources. Crucially, such centralized methods would not be able to scale in multi-cloud that consist of the Internet and public cloud data centers. Since they are highly dynamic in nature, we cannot collect all the required information and run an optimization algorithm frequently to keep up with the changes. Moreover, the scheduler would become a single point of failure.

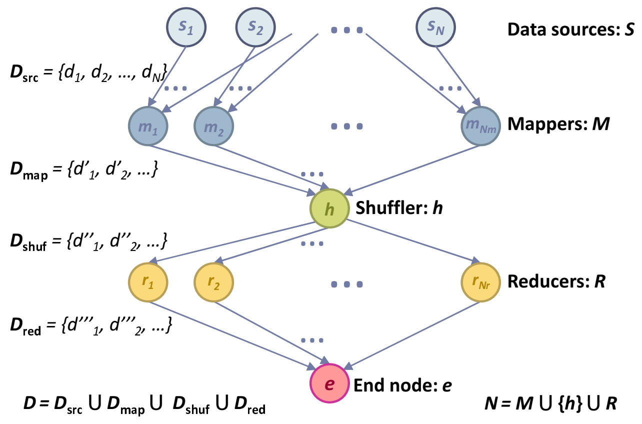
We plan to tackle these issues two-fold. First, we will formalize an optimization framework for MapReduce over multi-cloud and show that it is solvable with a centralized optimizer with limited scalability. Next, we will decentralize the formalized optimization algorithm, which has better scalability with compromised performance. The rest of the paper is organized as follows. Section 2 describes a formalization for a centralized MapReduce optimization over multi-cloud. Section 3 presents a brief concept of decentralized resource management for MapReduce. Finally, we conclude the paper in Section 4.

II. Centralized MapReduce Optimization over Multi-Cloud

In this section, we formalize a centralized optimization of MapReduce computation over multi-cloud.

1) Notations:

Nodes and data sets: The architecture of multi-cloud MapReduce is shown in Figure 1. $S = \{s_1, s_2, ..., s_{|S|}\}$ is a set of data sources. S collectively generates a source data set $D_{\mathrm{src}} = \{d_1, d_2, ..., d_{|S|}\}$, where d_i is produced by a source s_i. There are a set of *mappers* $M = \{m_1, m_2, ..., m_{|M|}\}$ to process D_{src}, which generates a set of mapped data $D_{\mathrm{map}} = \{d'_1, d'_2, ...\}$. D_{map} then is received by a *shuffler* h. All the mappers work in parallel, but the shuffling process does not finish until all the mapped data come to the shuffler. Once mapped data is shuffled, all the intermediate key-value pairs in D_{map} is grouped by key, the shuffler h outputs a set of shuffled data $D_{\mathrm{shuf}} = \{d''_1, d''_2, ...\}$ to *reducers* $R = \{r_1, r_2, ..., r_{|R|}\}$. Finally, the reducers process D_{shuf} and output a set of reduced data $D_{\mathrm{red}} = \{d'''_1, d'''_2, ...\}$ to the end node e. For notational simplicity, we call $N = M \cup \{h\} \cup R$ the entire set of processing nodes and $D = D_{\mathrm{src}} \cup D_{\mathrm{map}} \cup D_{\mathrm{shuf}} \cup D_{\mathrm{red}}$ the entire set of data.



s_1
s_2
s_N
Data sources: *S*
$D_{src} = \{d_1, d_2, ..., d_N\}$
m_1
m_2
m_{Nm}
Mappers: *M*
$D_{map} = \{d'_1, d'_2, ...\}$
h
Shuffler: *h*
$D_{shuf} = \{d''_1, d''_2, ...\}$
r_1
r_2
r_{Nr}
Reducers: *R*
$D_{red} = \{d'''_1, d'''_2, ...\}$
e
End node: *e*
$D = D_{src} \cup D_{map} \cup D_{shuf} \cup D_{red}$
$N = M \cup \{h\} \cup R$

Figure 1. High-level View of Multi-Cloud MapReduce.

For any data $d \in D$, the source and the destination nodes of d are given by the following functions: $f_{\mathrm{d \to n_s}}(d)$ defines the source node of data d and $f_{\mathrm{d \to n_d}}(d)$ defines the destination node of data d. Inversely, given a node $n \in N$, a set of data coming out of the node is given by the function $g_{\mathrm{n_s \to d}}(n) = f^{-1}_{\mathrm{d \to n_s}}(n)$. Similarly, a set of data coming into to the node is given by the function $g_{\mathrm{n_d \to d}}(n) = f^{-1}_{\mathrm{d \to n_d}}(n)$. Also, $f_{\mathrm{size}}(d)$ returns the size of data d.

Multi-cloud data centers: $C = \{c_1, c_2, ..., c_M\} \cup \{\bar{c}\}$ is a set of public cloud providers. A cloud provider has one or more regions that are geographically distributed. $P = \{\rho_1, \rho_2, ...\} \cup \{\bar{\rho}\}$ is a s set of regions. Likewise, a region has several zones internally. $Z = \{z_1, z_2, ...\} \cup \{\bar{z}\}$ is a set of zones, in which virtual machines (VMs) are launched. Relationships between C, P, and Z are defined by $f_{\mathrm{z \to \rho, c}} : Z \to P \times C$. We define special symbols $\bar{c}, \bar{\rho}$, and $\bar{z}$ to be used for VMs allocated outside the public clouds. For such VMs, $f_{\mathrm{z \to \rho, c}}(\bar{z}) = (\bar{\rho}, \bar{c})$. We intend to use these symbols to support VMs allocated in private clouds.

$\vec{t} = [t_1, t_2, ..., t_T]$ is a vector of available VM types, where each t_i is associated with a zone in a region from a cloud provider, the number of CPU cores $\gamma(t_i)$, and standardized computing power $\eta(t_i)$ such as defined in [11]. Even if there are multiple instance types with the same QoS and price, if they are in different regions, we treat them as different VM types. For example, `m3.medium` in US East and `m3.medium` in US West are two different VM types here.

$\pi(t_j)$ is a VM usage price for a VM type t_j per unit time. Inbound/outbound data transfer costs are defined for regions $\rho_i \in P$ $(i = 1, ..., |P|)$. $\pi^{\mathrm{in}}_{\mathrm{z}}(\rho_i), \pi^{\mathrm{in}}_{\rho}(\rho_i)$, and $\pi^{\mathrm{in}}_{\mathrm{inet}}(\rho_i)$ are inbound data transfer prices per unit data: $\pi^{\mathrm{in}}_{\mathrm{z}}(\rho_i)$ is for data transferred from another zone of the same region; $\pi^{\mathrm{in}}_{\rho}(\rho_i)$ is for data transferred from another region of the same cloud provider; and $\pi^{\mathrm{in}}_{\mathrm{inet}}(\rho_i)$ is for data transferred from the Internet. Note that $\pi^{\mathrm{in}}_{\mathrm{z}}(\bar{\rho}) = \pi^{\mathrm{in}}_{\rho}(\bar{\rho}) = \pi^{\mathrm{in}}_{\mathrm{inet}}(\bar{\rho}) = 0$. Outbound data transfer prices $\pi^{\mathrm{out}}_{\mathrm{z}}(\rho_i), \pi^{\mathrm{out}}_{\rho}(\rho_i)$, and $\pi^{\mathrm{out}}_{\mathrm{inet}}(\rho_i)$ are defined similarly: $\pi^{\mathrm{out}}_{\mathrm{z}}(\rho_i)$ is for data going to another zone of the same region; $\pi^{\mathrm{out}}_{\rho}(\rho_i)$ is for data going to another region of the same cloud provider; and $\pi^{\mathrm{out}}_{\mathrm{inet}}(\rho_i)$ is for data going to the Internet. Again, $\pi^{\mathrm{out}}_{\mathrm{z}}(\bar{\rho}) = \pi^{\mathrm{out}}_{\rho}(\bar{\rho}) = \pi^{\mathrm{out}}_{\mathrm{inet}}(\bar{\rho}) = 0$.

Virtual machine instances: We assume there can be at most $V_{\max}$ VM instances per instance type. VM instances are represented by a $V_{\max} \times T$ matrix V, where the (j, k)-th entry is $v_{jk} \in \{0, 1\}$. If $v_{jk} = 1$, we create a VM instance for the VM type t_k, we do not create any VMs otherwise. Clearly, $\sum_{1 \leqslant j \leqslant V_{\max}} v_{jk}$ is the total number of VM instances we create for the VM type t_k. Every processing node in N is mapped to a VM by a $|N| \times V_{\max} \times T$ matrix X. The (i, j, k)-th element of X is defined as $x_{ijk} \in \{0, 1\}$, where $x_{ijk} = 1$ if $n_i \in N$ is mapped to $v_{jk} = 1$, and $x_{ijk} = 0$ otherwise. To guarantee each node is mapped to only one VM, $\sum_{j=1}^{V_{\max}} \sum_{k=1}^{T} x_{ijk} = 1$ must hold for $i = 1, ..., |N|$. We assign a submatrix of X $(1 \leqslant i \leqslant |M|)$ for the mapping for the mappers, X $(i = |M| + 1)$ for the shuffler, and X $(|M| + 2 \leqslant i \leqslant |N|)$ for the reducer. We call them $\mathrm{X}_M, \mathrm{X}_h, \mathrm{X}_R$ respectively. Using x_{ijk}, we define a function $f_{\mathrm{n \to vm}}$ that maps nodes to VMs as follows. For $n_i \in N$, $f_{\mathrm{n \to vm}}(n_i) = v_{jk}$ s.t. $x_{ijk} = 1, v_{jk} = 1$. Every VM instance $v_{jk} \in V$ is mapped to a zone by the function $f_{\mathrm{vm \to z}}(v_{jk}) = z \in Z$ and a VM type by the function $f_{\mathrm{vm \to t}}(v_{jk}) = t_j$.

Networking parameters: For every arbitrary pair of VMs $v_{\mathrm{s}}, v_{\mathrm{d}} \in V$, $\lambda(v_{\mathrm{s}}, v_{\mathrm{d}})$ defines the latency between VMs v_{s} and v_{d}. Likewise, $\omega(v_{\mathrm{s}}, v_{\mathrm{d}})$ defines the bandwidth between VMs v_{s} and v_{d}.

Helper functions for data transfer cost computation: To compute data transfer costs, we define some more helper functions. For a node $n \in N$, $f_{\mathrm{n \to z}}(n) = (f_{\mathrm{vm \to z}} \circ f_{\mathrm{n \to vm}})(n)$ returns a zone where the node belongs to.

Using $f_{\mathrm{n \to z}}$ and $f_{\mathrm{z \to \rho, c}}$, for a source node n_{s} and a destination node n_{d}, we can obtain a triplet of a zone, a region, and a cloud provider as follows.

$\big(f_{\text{n}\to\text{z}}(n_{\text{s}}), f_{\text{n}\to\rho}(n_{\text{s}}), f_{\text{n}\to\text{c}}(n_{\text{s}})\big) = (z_{\text{s}}, \rho_{\text{s}}, c_{\text{s}})$ and similarly, $\big(f_{\text{n}\to\text{z}}(n_{\text{d}}), f_{\text{n}\to\rho}(n_{\text{d}}), f_{\text{n}\to\text{c}}(n_{\text{d}})\big) = (z_{\text{d}}, \rho_{\text{d}}, c_{\text{d}})$. Note that $f_{\text{n}\to\rho} = 1^{st} \circ f_{\text{z}\to\rho,\text{c}} \circ f_{\text{n}\to\text{z}}$ and $f_{\text{n}\to\text{c}} = 2^{nd} \circ f_{\text{z}\to\rho,\text{c}} \circ f_{\text{n}\to\text{z}}$, where 1^{st} and 2^{nd} are the functions to get the first and the second entries from a pair respectively (*i.e.*, $1^{st}(a,b) = a, 2^{nd}(a,b) = b$). For nodes outside the public cloud, we get a triplet $(\bar{z}, \bar{\rho}, \bar{c})$.

The following binary functions are defined to determine whether a data transfer from n_{s} to n_{d} is inter-zone, inter-region, or inter-cloud provider. Note that if the data transfer is local, *i.e.*, $c_{\text{s}} = c_{\text{d}}$ and $\rho_{\text{s}} = \rho_{\text{d}}$ and $z_{\text{s}} = z_{\text{d}}$, there is no cost associated with the data transfer.

$$b_{\text{z}}(n_{\text{s}}, n_{\text{d}}) = \begin{cases} 1 & \text{if } c_{\text{s}} = c_{\text{d}} \text{ and } \rho_{\text{s}} = \rho_{\text{d}} \text{ and } z_{\text{s}} \neq z_{\text{d}}, \\ 0 & \text{otherwise.} \end{cases}$$

$$b_{\rho}(n_{\text{s}}, n_{\text{d}}) = \begin{cases} 1 & \text{if } c_{\text{s}} = c_{\text{d}} \text{ and } \rho_{\text{s}} \neq \rho_{\text{d}}, \\ 0 & \text{otherwise.} \end{cases}$$

$$b_{\text{c}}(n_{\text{s}}, n_{\text{d}}) = \begin{cases} 1 & \text{if } c_{\text{s}} \neq c_{\text{d}}, \\ 0 & \text{otherwise.} \end{cases}$$

2) Processing Time and Cost:

Processing time: Suppose we are given functions to compute processing times: $\tau_{\text{map}}(m_i, d_j, v_{m_i})$ for a mapper m_i to process data d_j on the mapped VM v_{m_i}, $\tau_{\text{shuf}}(h, d'_j, v_h)$ for the shuffler h to process data d'_j on the mapped VM v_h, and $\tau_{\text{red}}(r_i, d''_j, v_{r_i})$ for a reducer r_i to process data d''_j on the mapped VM v_{r_i}, we can compute processing times for the map, shuffle, and reduce phases as follows. Note that v_{m_i}, v_h, and v_{r_i} are given by $f_{\text{n}\to\text{vm}}(m_i), f_{\text{n}\to\text{vm}}(h)$, and $f_{\text{n}\to\text{vm}}(r_i)$ respectively.

$$\begin{aligned} t_{\text{map}}(V, \text{X}_M) &= \max_{m_i \in M} \sum_{d_j \in g_{\text{n}_{\text{d}}\to\text{d}}(m_i)} \Bigg[\frac{f_{\text{size}}(d_j)}{\omega(v_s, v_{m_i})} + \lambda(v_s, v_{m_i}) \\ &+ \tau_{\text{map}}(m_i, d_j, v_{m_i}) + \frac{f_{\text{size}}(d'_j)}{\omega(v_{m_i}, v_h)} + \lambda(v_{m_i}, v_h) \Bigg], \end{aligned}$$

where $v_s = f_{\text{d}\to\text{n}_{\text{s}}}(d_j)$, d'_j represents processed d_j by m_i, and $v_s = f_{\text{n}\to\text{vm}}(s)$.

$$\begin{aligned} t_{\text{shuf}}(V, \text{X}_h) &= \tau_{\text{shuf}}(h, d'_j, v_h) \\ t_{\text{red}}(V, \text{X}_R) &= \max_{r_i \in R} \sum_{d''_j \in g_{\text{n}_{\text{d}}\to\text{d}}(r_i)} \Bigg[\frac{f_{\text{size}}(d''_j)}{\omega(v_h, v_{r_i})} + \lambda(v_h, v_{r_i}) \\ &+ \tau_{\text{red}}(r_i, d''_j, v_{r_i}) + \frac{f_{\text{size}}(d'''_j)}{\omega(v_{r_i}, v_e)} + \lambda(v_{r_i}, v_e) \Bigg], \end{aligned}$$

where d'''_j represents processed d''_j by r_i and $v_e = f_{\text{n}\to\text{vm}}(e)$. t_{map} takes the maximum time over all processing times from the mappers in M. Each mapper m_i takes the time to: 1) receive source data from a source, 2) process the data at the mapper, and 3) transfer the processed data to the shuffler. Similarly, t_{red} takes the maximum time over all processing times from the reducers in R. The total processing time can be represented as $t_{\text{total}}(V, X) = t_{\text{map}} + t_{\text{red}}$. Note that we do not include t_{shuf} because the shuffler h can process individual mapped data incrementally as they arrive from the mappers. Assuming the shuffler has enough computing power, the time for shuffling is masked by t_{map}.

Cost: The total VM usage cost π_{vm} is given as follows:

$$\begin{aligned} \pi_{\text{vm}}(V, X) &= \sum_{j=1}^{V_{\max}} \sum_{k=1}^{T} \pi(f_{\text{vm}\to\text{t}}(v_{jk})) \times \\ &\Bigg[t_{\text{map}} \cdot H\Big(\sum_{i=1}^{|M|} x_{ijk}\Big) + t_{\text{shuf}} \cdot x_{|M|+1,jk} + \\ &t_{\text{red}} \cdot H\Big(\sum_{i=1}^{|R|} x_{|M|+1+i,jk}\Big) \Bigg], \end{aligned}$$

where $H(x)$ returns 1 only if $1 \leqslant x$, otherwise 0. The data transfer cost π_{data} is given as follows:

$$\pi_{\text{data}}(V, X) = \sum_{d \in D} f_{\text{size}}(d) \times \pi_{\text{d}}(n_{\text{s}}, n_{\text{d}}),$$

where $n_s = f_{\text{d}\to\text{n}_{\text{s}}}(d)$, $n_d = f_{\text{d}\to\text{n}_{\text{d}}}(d)$ and

$$\begin{aligned} \pi_{\text{d}}(n_{\text{s}}, n_{\text{d}}) &= b_{\text{z}}(n_{\text{s}}, n_{\text{d}})\Big((\pi_{\text{z}}^{\text{out}} \circ f_{\text{n}\to\rho})(n_{\text{s}}) + (\pi_{\text{z}}^{\text{in}} \circ f_{\text{n}\to\rho})(n_d)\Big) \\ &+ b_{\rho}(n_{\text{s}}, n_{\text{d}})\Big((\pi_{\rho}^{\text{out}} \circ f_{\text{n}\to\rho})(n_{\text{s}}) + (\pi_{\rho}^{\text{in}} \circ f_{\text{n}\to\rho})(n_d)\Big) \\ &+ b_{\text{c}}(n_{\text{s}}, n_{\text{d}})\Big((\pi_{\text{c}}^{\text{out}} \circ f_{\text{n}\to\rho})(n_{\text{s}}) + (\pi_{\text{c}}^{\text{in}} \circ f_{\text{n}\to\rho})(n_d)\Big). \end{aligned}$$

Notice that $f_{\text{n}\to\rho}$ in π_{d} uses $f_{\text{n}\to\text{vm}}$, so we can see π_{data} as a function of V and X. Finally, the total cost $\pi_{\text{total}}(V, X) = \pi_{\text{vm}} + \pi_{\text{data}}$. π_{data} is a summation of data transfer costs for all data $d \in D$. If a data transfer is either inter-zone, inter-region, or inter-cloud provider, then data transfer is charged for both inbound and outbound by one of the binary functions b_{z}, b_{ρ}, and b_{c}. Note that if a node is not in the public cloud, then $f_{\text{n}\to\rho}$ returns $\bar{\rho}$, which is not subject to charge.

3) Problem Formulations:

Given the information from Section II-1 and Section II-2, we define both time and budget-constrained non-linear integer programming problems of MapReduce over multi-cloud as follows:

- **Time-constrained cost minimization**:

$$\begin{aligned} \min \quad & \pi_{\text{total}}(V, X), \\ \text{subject to} \quad & t_{\text{total}}(V, X) \leqslant t_{\text{deadline}}, \\ & x_{ijk} \in \{0,1\}, v_{jk} \in \{0,1\}, \\ & \sum_{j=1}^{V_{\max}} \sum_{k=1}^{T} x_{ijk} = 1, \forall i \in [1, |N|], \quad (1) \\ & H(\sum_{i=1}^{|N|} x_{ijk}) = v_{jk}, \quad (2) \\ & \quad \forall j \in [1, V_{\max}], \forall k \in [1, T]. \end{aligned}$$

  where t_{deadline} is a given time deadline constraint.
- **Budget-constrained time minimization**: Given a budget constraint π_{budget}, minimize $t_{\text{total}}(V, X)$ subject to $\pi_{\text{total}}(V, X) \leqslant \pi_{\text{budget}}$. Other constraints are given by Equation 1 and 2.

The first constraint (Equation 1) is to ensure each node is mapped to only one VM instance. The second constraint

(Equation 2) is to ensure that there is at least one node mapped to a created VM, and that no nodes are mapped to VMs that do not exist.

Both t_{total} and π_{total} are non-linear and the above non-linear integer programming problems are NP-hard [12]. Thus, there is no known algorithm obtaining the optimal solution to these problems efficiently. We are now working to find heuristics (*e.g.*, Particle Swarm Optimization [13]) that give cost-efficient high-performance solutions within a reasonable amount of time.

III. Towards Decentralized Internet-Scale Data Analytics

Here, we describe the basic idea of decentralized resource management for MapReduce applications. As shown in Figure 2, there are three layers in the framework: the VM resource layer, the agent network (middleware) layer, and the application layer. There is an agent per VM that independently makes application resource reconfiguration decisions.

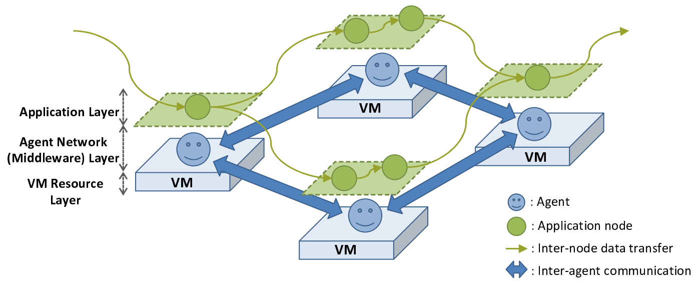

Application Layer
Agent Network (Middleware) Layer
VM Resource Layer
VM
VM
VM
VM
: Agent
: Application node
: Inter-node data transfer
: Inter-agent communication

Figure 2. Layered architecture of the decentralized resource management framework.

Given a time or budget constraint from the user, the goal of decentralized resource management is to find a resource configuration that is efficient in both cost and performance while satisfying the given constraint. Whether the application is satisfying the constraint (*i.e.*, throughput or cost per time) or not, each agent always tries to improve the throughput by migrating one of the application nodes to another VM. Only if the application is not satisfying the constraint, the agent creates a new VM that satisfies the constraint and migrates one of the application nodes to the newly created VM. Before an actual migration is performed, it is important for the agent to predict the performance after the migration. We are currently making progress on the performance prediction model.

IV. Conclusion

We are working towards creating a scalable Internet-scale data analytics framework over multi-cloud. In this paper, as a first step to achieve that goal, we first formalized an optimization framework for MapReduce over multi-cloud, and then we showed a brief idea of decentralized resource management middleware that considers both cost and performance during resource allocation.

Acknowledgment: The authors would like to thank an Amazon Web Services educational research grant and a Google Cloud Credits Award.

References

[1] D. Petcu, “Multi-cloud: expectations and current approaches,” in *Proceedings of the 2013 International Workshop on Multi-cloud applications and federated clouds*. ACM, 2013, pp. 1–6.

[2] A. N. Toosi, R. N. Calheiros, and R. Buyya, “Interconnected cloud computing environments: Challenges, taxonomy, and survey,” *ACM Computing Surveys (CSUR)*, vol. 47, no. 1, p. 7, 2014.

[3] J. Dean and S. Ghemawat, “MapReduce: simplified data processing on large clusters,” *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[4] The Apache Software Foundation, “Apache Hadoop,” http://hadoop.apache.org/, Accessed: 02-10-2015.

[5] L. Wang, J. Tao, R. Ranjan, H. Marten, A. Streit, J. Chen, and D. Chen, “G-Hadoop: MapReduce across distributed data centers for data-intensive computing,” *Future Generation Computer Systems*, vol. 29, no. 3, pp. 739–750, 2013.

[6] A. Iordache, C. Morin, N. Parlavantzas, E. Feller, and P. Riteau, “Resilin: elastic MapReduce over multiple clouds,” in *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*. IEEE, 2013, pp. 261–268.

[7] A. Mandal, Y. Xin, I. Baldine, P. Ruth, C. Heerman, J. Chase, V. Orlikowski, and A. Yumerefendi, “Provisioning and evaluating multi-domain networked clouds for Hadoop-based applications,” in *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*. IEEE, 2011, pp. 690–697.

[8] Q. Zhang, L. Liu, K. Lee, Y. Zhou, A. Singh, N. Mandagere, S. Gopisetty, and G. Alatorre, “Improving Hadoop service provisioning in a geographically distributed cloud,” in *Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on*. IEEE, 2014, pp. 432–439.

[9] K. Chen, J. Powers, S. Guo, and F. Tian, “CRESP: Towards optimal resource provisioning for MapReduce computing in public clouds,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 6, pp. 1403–1412, 2014.

[10] F. Tian and K. Chen, “Towards optimal resource provisioning for running MapReduce programs in public clouds,” in *Cloud Computing (CLOUD), 2011 IEEE International Conference on*. IEEE, 2011, pp. 155–162.

[11] S. Imai, T. Chestna, and C. A. Varela, “Accurate resource prediction for hybrid IaaS clouds using workload-tailored elastic compute units,” in *Proceedings of the 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*. IEEE Computer Society, 2013, pp. 171–178.

[12] W. Murray and K.-M. Ng, “An algorithm for nonlinear optimization problems with binary variables,” *Computational Optimization and Applications*, vol. 47, no. 2, pp. 257–288, 2010.

[13] M. A. Rodriguez and R. Buyya, “Deadline Based Resource Provisioning and Scheduling Algorithm for Scientific Workflows on Clouds,” *IEEE Transactions on Cloud Computing*, vol. 2, no. 2, pp. 222–235, 2014.