Live Data Migration Strategy with Stream Processing Framework

KUN MA, QIUCHEN CHENG, and BO YANG
University of Jinan
Shandong Provincial Key Laboratory of Network Based Intelligent Computing
336 West Road of Nan Xinzhuang, Jinan 250022
CHINA
ise_mak@ujn.edu.cn, chengqc.m@gmail.com, yangbo@ujn.edu.cn

Abstract: Live data migration in the cloud is responsible to migrate blocks of data of emigration node to several immigration node. However, live data migration strategy is a NP-hard problem like task scheduling. Recently, in-stream processing is the immediate need in many practical applications. Therefore, we explore a real-time live data migration strategy with stream processing framework in this paper. First, the migration cost and balance model is introduced as the metrics to evaluate data migration strategy. Subsequently, a live data migration strategy with particle swarm optimization is proposed. Afterwards, we implement this method using stream processing framework. The experimental results show the best performance of our method in all.

Key–Words: load balancing, stream processing, data migration, particle swarm optimization

1 Introduction

Cloud computing [1], as a new computing paradigm, has recently received considerable attention in both academic and industry to provide dynamically scalable and virtualized resource (such as infrastructure, software and data) as a service over the Internet. By this means, users will be able to access large-scale data in the cloud anywhere and anytime on demand. Cloud elasticity [2], the ability to use as many resources as needed with low cost at any given time, the service provider must take load balancing of the data resources into consideration. For example, the cloud nodes providing data access services need dynamic data migration to guarantee that all the data nodes can provide stable service [3] [4]. The data nodes with heavy load caused by the emergency and uncertainty of data access will become the bottleneck of data services. Therefore, the requirement of live data migration strategy in the cloud is urgent to guarantee quality of service and satisfy the elasticity of cloud computing.

Live data migration in the cloud means that the data resource of each node is responsive all the time during the migration process from the clients' perspective. Compared with traditional suspend/resume migration, live migration holds many benefits such as energy saving, load balancing, and online maintenance [5]. Many live migration strategy methods are proposed to improve the migration efficiency. The first solution is making migration plan by migration cost [6] [5]. A load balancing framework, supporting a wide range of measures for load imbalance and reconfiguration cost, are proposed to reconfiguring a system facing dynamic workload changes [6]. To improve the migration strategy, greedy algorithm is used to determine the best migration solution [5]. The second solution is making migration plan by hot-spot neighbor [7] [8]. The hot-spot data will be migrated to the nearest neighbor by some metrics [7]. A subset of objects from hot-spot servers is selected to perform topology-aware migration to minimize reconfiguration costs [8]. Some more intelligent algorithm are assisted to make a migration strategy. The frequency of data access and block size of migration data are considered to find target nodes using particle swarm optimization algorithm [9]. More metrics such as priority of tenant ant its data are considered in the improved method [10].

Live data migration strategy is a NP-hard problem like task scheduling [11]. Therefore, particle swarm optimization (PSO) and genetic algorithm (GA) are used to address this issue. Both algorithms are based on collaborative population-based search, but the particle swarm optimization is better in efficiency [12] [13]. Reference [14] gives more analysis on how particle swarm optimization benefits task assignment. Discrete particle swarm optimization (DPSO) is also used to address load balancing [15] [16] [17]. Although traditional PSO is effective, each particle that is computing in serial will cause too much waste in case of big data. Recently, several researchers proposed to improve PSO in parallel such as MapReduce framework [18] [19] [20], where experiment results

show that PSO with MapReduce is faster. However, this batch processing framework need repeating start-up of migration strategy job.

Recently, the shortcomings and drawbacks of batch-oriented data processing (e.g. MapReduce framework) were widely recognized by the Big Data community. In-stream processing [21] [22] is the immediate need in many practical applications. Therefore, it is natural that we wonder how both live data replication strategy and stream processing framework can be integrated and benefit from each other. In this paper, we first introduce the migration cost and imbalance model as the metrics to evaluate data migration strategy. Then, a live data migration strategy with PSO is proposed. Afterwards, we implement this approach using stream processing framework. Finally, live data replication approach with stream processing framework shows the best performance of our method in all.

The rest of the paper is organized as follows. Section 2 discusses the related work. In Section 3, we introduce the methodology of live data migration strategy with stream processing framework. The migration cost and imbalance model is firstly introduced to evaluate the performance of the data migration. Subsequently, this live data migration strategy using PSO is proposed, and then implemented with the stream processing framework. Section 4 gives the experimental evaluation of this live data migration strategy. Brief conclusions are outlined in the last section.

2 Related Work

2.1 Data Migration Strategy

With the emerging of cloud computing and batch computing, there are mainly two types of classical data migration strategies: online live migration [23] and offline batch-oriented migration. Online live migration continuously analyzes the load in motion to deliver real-time analytic in real time, and migrates the data from the node with heavy load to the one with light load. Offline batch-oriented migration is another way to migrate the data in parallel. Generally, MapReduce framework [24] is selected to implement this method.

There are mainly two types of skews for data usage in the cloud: data skew and load skew. One or both of them might exist in the system anytime. Great efforts have been paid to deal with data skew to guarantee a uniform distribution of data among cluster nodes in the previous work [25] [26]. Whereas the rebalancing is also costly, load skew is the main focus of this paper, which is more likely to cause nodes to be overloaded. We assume that this is the main factor causing imbalance, and other imbalances can be easily tolerated.

2.2 Straightforward Method

Live data migration is in charge of transferring blocks of data from emigration nodes to immigration nodes. The data are stored in the form of blocks, and each block contains several records. Therefore, the data migration problem means migrating m blocks of data to different target n nodes. As mentioned above, the data migration problem can be generalized to task scheduling problem [27] or binpacking problem [28], which is NP hard. There are several intelligent method to address this issue, such as particle swarm optimization (PSO), genetic algorithm (GA), and greedy algorithm. A modified PSO-based task scheduling method was proposed to reallocate migrated virtual machines in the overloaded host and dynamically consolidate the under-loaded host [29]. Skewness and refined cost model are improved to address resource allocation of virtual resources in the cloud later [30] [27]. Some researchers investigated to use PSO to address load balancing problem [31] [15].

After we evaluate the cons and pros of the above mentioned methods, we select PSO to determine how to migrate data from emigration nodes to immigration nodes. Compared with straightforward methods, we attempt to implement live data migration strategy in Parallel.

2.3 Parallel Method

Several modern parallel methods have been proposed to address binpacking-like problem in order to improve the performance using MapReduce techniques. PSO with MapReduce framework is proposed to address intrusion detection and overlay network optimization [32] [33].

3 Methodology of Live Data Migration Strategy

3.1 Migration Cost Model and Balance Model

3.1.1 Migration Cost Model

Migration cost is the trade-offs between the migration time and performance impact. We represent the migration cost model in a two-dimensional space, where x axis denotes migration time t and y axis denotes performance impact i. The migration cost of at any given moment is represented by the product of migration

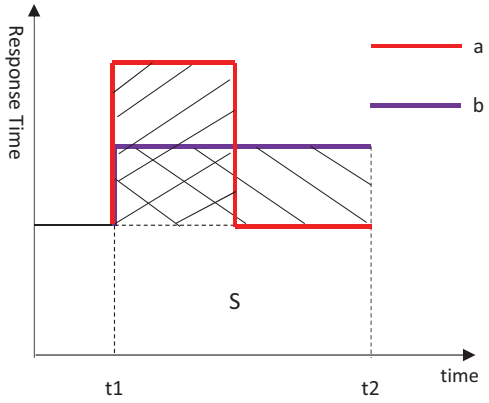


Figure 1: Migration cost model.

time and performance impact. The cost of a migration is represented by the a given rectangle (computed as: $cost$=$(t2-t1)$*$(i2-i1)$) (see Figure 1). Consequently, the trade-off problem is converted into the rectangle selection problem. In fact, this rectangular area-based cost model is built based on the idea of integration in mathematics. Given a function of response time and an interval of time, the definite integral could be computed. Figure 1 shows the response time in 2 migrations (a and b). Consider the longest interval (from $t1$ to $t2$) as the basis. The average response time $\overline{r}$ can be represented as $\overline{r} = \int_{t1}^{t2} f(t)dt/(t2-t1)$, where $f(t)$ is the instant response time. The integration can be regarded as the shadow area S_s plus the blank area S_b. Since the blank area S_b is the same with different migrations, the average response time is finally decided by the shadow area S_s.

3.1.2 Balance Model

Load skew is the main factor that affects the performance of the data node. Next, we propose a balance model of the data nodes. Consider l_i be the load value on node i, then the normalized load value is $p_i = l_i/\sum_{i=1}^{n}(l_i)$. According to Shannon's theory [34], the information entropy may serve as a measure of mix-up of a distribution. Thus we build the balance function F based on Shannon's theory. The entropy of P is denoted as Equation 1.

$$H(p) = -\sum_{i=1}^{n}(p_i * logp_i) \tag{1}$$

Obviously, the larger the entropy is, the more balancing the load is. It is clear that the maximum value of $H(p)$ is log(n) if and only if $p_i = 1/n$, which corresponds to the most uniform load distribution. Afterwards, the balance model is represented by the normalized entropy as Equation 2.

$$F = H(P)/log(n) = -\sum_{j=1}^{n}(P_j * logP_j)/log(n) \tag{2}$$

3.2 Live Data Migration Strategy with Discrete Particle Swarm Optimization

3.2.1 Problem Description

In fact, live data migration problem means migrating m blocks of data of 1 source emigration node to n target immigration nodes. The constraint criteria is that 1 block of data can only migrated into 1 immigration node, that is called 1 migration task, and all the tasks can executed in parallel. The objective of live data migration strategy is the minimum value of the quotient of migration cost C and loading balance H. Migration cost C is calculated as the migration cost model shown in Equation 1, denoted as $C = T * I$, where migration time T depends on the number of blocks od data, and performance impact I depends on the loading performance of the immigration node. Loading balance of migration cost H is calculated by the normalized entropy of loading balance shown in Equation 2.

Live data migration strategy is described as how to migrate m blocks of data of 1 source emigration node to n target immigration nodes. Consider D_i (1¡=i¡=m) as the size of block i, P_j (1¡=j¡=n) is the loading performance of the immigration node j. The output of live data migration strategy is x_{ij}, where $x_{ij} = 1$ means migrating block i to immigration node j. Besides, the fitness of the quotient of migration cost and its loading balance is also calculated.

3.2.2 Fitness

The fitness of live data migration strategy is considered as the quotient of migration cost and its loading balance, denoted as Equation 3. T is the migration time (represented by size of block D_i), I is the performance impact (represented by load of immigration node P_j), and H is normalized entropy of migration cost.

$$\begin{aligned} F &= \frac{C}{H} = \frac{T * I}{H} \\ &= \frac{\sum_{i=1}^{m}\sum_{i=1}^{n}(X_{ij} * Tij) * \sum_{i=1}^{m}\sum_{i=1}^{n}(x_{ij} * Iij)}{-\sum_{j=1}^{n}(x_{ij} * (Tij * Iij) * log(Tij * Iij))/log(n)} \\ &= \frac{\sum_{i=1}^{m}\sum_{i=1}^{n}(X_{ij} * D_i) * \sum_{i=1}^{m}\sum_{i=1}^{n}(x_{ij} * P_j)}{-\sum_{j=1}^{n}(x_{ij} * (D_i * P_j) * log(D_i * P_j))/log(n)} \end{aligned} \tag{3}$$

3.2.3 Position and Velocity

In the problem of live data migration strategy, the position of a particle is a feasible migration task, migrating a block of data to a target immigration node. The position matrix X is denoted as $X = \{x_{ij}\}$ (1¡=i¡=m, 1¡=j¡=n). The value of x_{ij} is either 0 or 1. $x_{ij} = 1$ means block i is migrated into immigration node j. The constraint criteria of x_{ij} is $\sum_{j=1}^{n} x_{ij} = 1$. The velocity matrix of a particle $V =$ is denoted as Equation 4, where v_{ij} is the velocity of a particle, w is the inertia weight, c_1 and c_1 are learning factors respectively, r_1 and r_1 are random values in range [0, 1], pb is the previous best fitness value of this particle, and gb is the global best fitness value. The velocity value of each particle is confined within [-vmax,vmax].

$$v_{ij}^{t+1} = w * v_{ij}^{t} + c_1 * r_1 * (pb - x_{ij}) + c_2 * r_2 * (gb - x_{ij}) \quad (4)$$

Since only 1 particle in the same row of the position matrix is 1, we update the position of a particle according the maximum value of sigmoid function of velocity.

$$x_{ij} = \begin{cases} 1 & if sigmod(v_{ij}) = max_{j=1}^{n}(sigmod(v_{ij})), \forall i \in \{1, 2, ..., m\} \\ 0 & otherwise \end{cases}$$

In order to avoid all the blocks are migrated into the same immigration node with small load performance, we propose a safeguard measure to guarantee x_{ij} is not equal to $x_{(i+1)j}$. If x_{ij}=$x_{(i+1)j} = 1$, we set $x_{(i+1)j}$ $as 0$, and choose another particle with the second largest $sigmod(v_{ij})$ to set as 1.

3.2.4 Algorithm

The pseudo code of the proposed PSO algorithm to address live data migration strategy is stated as Algorithm 1:

Algorithm 1 Live data migration strategy using PSO *pso*

Require: block size array of data in the emigration node D; load performance array of immigration node P;

Ensure: particle position matrix X; global fitness F;

```
initialize velocity matrix;
initialize position matrix (0 or 1);
normalize D and P;
initialize pbest pb and gbest gb
while result is not convergent do
    compute fitness using Eq. 3;
    update pbest pb;
    update velocity using Eq. 4;
    update position;
    update gbest gb;
end while
endprocedure
```

3.3 Live Data Migration Strategy with Stream Processing Framework

3.3.1 Stream Topology

In order to implement the real-time decision to migrate blocks of data to immigration nodes, we attempt to propose a stream processing framework. To do real-time computation, the stream topology is present first, which is shown in Figure 2. A topology is a graph of computation of live data migration strategy. Each computation node in a topology contains processing logic, and links between nodes indicate how in-stream

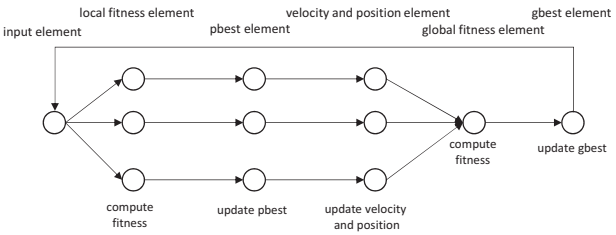


Figure 2: Stream topology of live data migration strategy.

data should be passed around between nodes. The input of blocks of data of emigration nodes and load performance of immigration nodes are called stream in the stream processing framework. A stream is abstracted as an unbounded sequence of tuples. This stream processing framework provides the primitives for transforming a stream into a new stream in a distributed and reliable way. Finally, the stream after the last processing element is the output strategy of live data migration. Networks of processing element are packaged into a topology which is the top-level abstraction. A topology is a graph of stream transformations where each node is a processing element. Edges in the graph indicate which processing elements are subscribing to which streams. When a processing element emits a tuple to a stream, it sends the tuple to every processing element that subscribed to that stream.

3.3.2 Processing Element

Processing element provides for doing stream transformations to migrate blocks of data to immigration n-

odes. This processing element have interfaces to help developers implement the application-specific logic to complete the migration. A source of streams of the input element is emitted to the stream processing framework, then grouped to compute fitness, update pbest, velocity and position of each particle. A input element reads tuples of block size of data of emigration nodes and load performance of immigration nodes, and emit them as a stream. The subsequent processing element of the stream processing framework consumes any number of input streams to implement PSO processing, and possibly emits new streams. Complex stream transformations require multiple steps and processing elements, to generate the PSO result.

Several grouping tasks to compute fitness, update pbest, velocity and position of each particle are in parallel. Local fitness element calculate the fitness of particles in this group according to Equation 3, pbest element means updating pbest, velocity and position element is responsible to update velocity and position according to Equation 4. Afterwards, all the particles are put together to compute global fitness element and update gbest.

3.3.3 Grouping Strategy

In order to make PSO execution in parallel as many tasks across the cluster and reduce the convergence rate of PSO, we group particles before computing fitness. The particles in different groups evolve independently. We adopt fitness grouping strategy to partition particles, which answers this question by telling stream processing framework how to send tuples between sets of tasks. According to the global fitness, we partition particles into several groups.

3.3.4 Fault Tolerance

We provide one method that is called implicit lineage tracking to deal with fault tolerance. Each in-stream compuation is guaranteed to be processed at least once by verifying the transactional unique ID. In case of failure, the PSO computation is recovered from the checking point.

We provide an acknowledgment task to track the directed acyclic graph (DAG) of tuples for every tuple of input element. When this task sees that a DAG is complete, it sends a message to the input element that created the acknowledgment message. When a tuple is created in a topology, it is given a random 64 bit ID within its lifecycle. These IDs are used by acknowledgment tasks to track the tuple DAG for every tuple of input element. Every tuple knows the ids of all the tuples of input element for which it exists in their tuple trees. After emitting a new tuple in the intimidate processing element, the tuple ids of input element from the tuple's anchors are copied into the new tuple. When a tuple is acknowledged, it sends a message to the corresponding acknowledgment tasks with information about how the tuple tree changed.

Acknowledgment tasks do not track the tree of tuples explicitly. For large tuple trees with tens of thousands of nodes, tracking all the tuple trees could overwhelm the memory. Instead, the acknowledgment tasks take a XOR strategy that only requires a fixed amount of space per tuple. An acknowledgment task stores a map from a tuple id to a pair of values. The first value is the task id creating the tuple of input element, which sends completion messages later. The second value is a 64 bit number called the acknowledgment value, which is a representation of the state of the entire tuple tree. It is simply the XOR of all tuple ids that have been created and/or acknowledged in the tree. When an acknowledgment task sees that an acknowledgment value has become 0, then it knows that the tuple tree is completed. Since tuple ids are random 64 bit numbers, the chances of an acknowledgment value accidentally becoming 0 is extremely small.

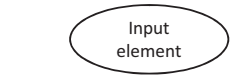
As shown in Figure 3(a), the initial value of acknowledgment is $1A0F$. After emitting tuples to processing element 1, the acknowledgment value is $1A0F$ xor $1A0F$ (see Figure 3(b)). Next, the processing element 1 emits two groups of tuples to processing element 2 and 3, the acknowledgment value is $1AoF$ xor $1A0F$ xor 2007 xor $300A$ (see Figure 3(c)). Finally, processing element 2 and 3 generates tuples to the final processing element, the acknowledgment value becomes $1AoF$ xor $1A0F$ xor 2007 xor $300A$ xor 2007 xor $300A$=0. The acknowledgment value 0 indicates that this task has completed successfully.

With the above methods, the stream processing framework avoids data loss in case of failure.

- When a tuple is not acknowledged due to the died task, the tuple IDs of input element at the root of the trees for the failed tuple will time out and be replayed.

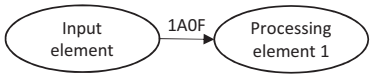
- When the acknowledgment task dies, all the tuples of input element that the acknowledgment task was tracking will time out and be replayed.

- When the task of input element dies, the data source that the spout talks to is responsible for replaying the messages.



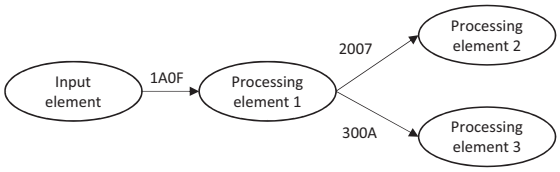
acknowledgment value: 1A0F

(a) Initial acknowledgment value.



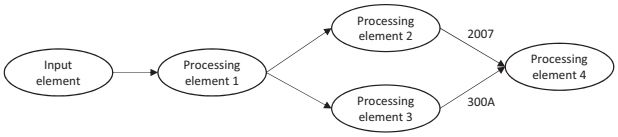
acknowledgment value: 1A0F xor 1A0F

(b) Acknowledgment value after emitting tuples to processing element 1.



acknowledgment value: 1AoF xor 1A0F xor 2007 xor 300A

(c) Acknowledgment value after processing element 1.



acknowledgment value: 1AoF xor 1A0F xor 2007 xor 300A xor 2007 xor 300A=0

(d) Final acknowledgment value.

Figure 3: Acknowledgment task is updating acknowledgment value

4 Experiment Results

We have made two experiments to evaluate our approach to make a decision of live data migration. The first experiment is to evaluate the grouping improvement of PSO, and the second experiment is to evaluate the instantaneity of PSO with with stream processing framework. The evaluation indicators is the ratio between migration cost and its balance, which is shown in Equation 3.

4.1 Standard PSO vs. proposed PSO

This experiment is to illustrate the superiority of our proposed PSO. We run our experiments an Intel Core(TM) i5-2300 @2.8 GHz CPU, 8GB memory and runs a 64-bit CentOS Linux OS with a Java 1.6 64-bit server JVM. The fitness and execution time of standard PSO and our proposed PSO are shown in Table 1. As evident, the proposed PSO performs better than classical PSO, especially in case of large amount of blocks of data and immigration nodes.

4.2 PSO with MapReduce vs. PSO with Stream Processing Framework

This experiment is to illustrate the instantaneity of PSO with stream processing framework. We use Hadoop 2.2 and Apache Storm 0.95 to run MapReduce and stream topology. We implement our proposed PSO with Hadoop MapReduce and Apache Storm respectively. The MapReduce job of our proposed PSO is assgined with 6 map and 3 reduce tasks, and speculative execution feature of Hadoop is disabled. Table 2 shows the fitness and execution time of PSO with MapReduce and PSO with stream processing framework. The execution time of PSO with stream processing framework is slightly faster than PSO with MapReduce. Without repeated startup of PSO jobs, PSO with stream processing framework is better in in-stream live data migration strategy overall.

5 Conclusions

There are many methods to address live data migration strategy, but most of them mainly focus on the parallel execution. For the in-stream strategy, batch processing is not always a good way. Therefore, we investigate another stream processing method to address live data migration strategy. Both migration cost and balance models are proposed to evaluate the metrics of migration strategy. Next, the live data migration strategy using PSO and stream processing frame-

Table 1: Fitness and execution time of standard PSO and proposed PSO.

No.	Blocks of data	Immigration nodes	Iteration	PSO		Grouping PSO	
				Fitness	Execution time (ms)	Fitness	Execution time (ms)
1	500	100	500	8.34	3999	8.22	3819
2	1000	100	500	5.32	8031	4.66	7551
3	3000	100	500	2.07	39872	1.31	22466
4	500	300	1000	3.12	12341	2.85	11181
5	1000	300	1000	2.35	23123	1.29	22317
6	3000	300	1000	4.39	73123	3.98	66679

Table 2: Fitness and execution time of PSO with MapReduce and PSO with stream processing framework.

No.	Blocks of data	Immigration nodes	Iteration	PSO with MapReduce		PSO with stream processing fram	
				Fitness	Execution time (ms)	Fitness	Execution time (ms)
1	500	100	500	8.19	774	8.23	647
2	1000	100	500	4.59	1510	4.53	1259
3	3000	100	500	1.39	5503	1.41	3754
4	500	300	1000	2.84	2478	2.72	1874
5	1000	300	1000	1.37	4437	1.31	3730
6	3000	300	1000	4.09	13345	3.95	11123

work are present to illustrate the instantaneity of migration.

Acknowledgements: This work was supported by the National Key Technology R&D Program (2012BAF12B07), the Shandong Provincial Natural Science Foundation (ZR2014FQ029), the Shandong Provincial Key R&D Program (2015GGX106007), the Doctoral Fund of University of Jinan (XBS1237), the National Training Program of Innovation and Entrepreneurship for Undergraduates (201410427030), and the Teaching Research Project of University of Jinan (J1344).

References:

[1] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

[2] Paul C Brebner. Is your cloud elastic enough?: performance modelling the elasticity of infrastructure as a service (iaas) cloud applications. In *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*, pages 263–266. ACM, 2012.

[3] Sudipto Das, Shoji Nishimura, Divyakant Agrawal, and Amr El Abbadi. Albatross: lightweight elasticity in shared storage databases for the cloud using live data migration. *Proceedings of the VLDB Endowment*, 4(8):494–505, 2011.

[4] Maryam Razavian and Patricia Lago. A lean and mean strategy: a data migration industrial study. *Journal of Software: Evolution and Process*, 26(2):141–171, 2014.

[5] Xiulei Qin, Wenbo Zhang, Wei Wang, Jun Wei, Xin Zhao, and Tao Huang. Towards a cost-aware data migration approach for key-value stores. In *Cluster Computing (CLUSTER), 2012 IEEE International Conference on*, pages 551–556. IEEE, 2012.

[6] Daniel Kunkle and Jiri Schindler. A load balancing framework for clustered storage systems. In *High Performance Computing-HiPC 2008*, pages 57–72. Springer, 2008.

[7] Flavio Pfaffhauser. *Scaling a cloud storage system autonomously*. PhD thesis, Masters thesis, ETH Zuerich, 2010.

[8] Gae-won You, Seung-won Hwang, and Navendu Jain. Scalable load balancing in cluster storage systems. In *Proceedings of the 12th International Middleware Conference*, pages 100–119. International Federation for Information Processing, 2011.

[9] Xiaojun Ren, Yongqing Zheng, and Lanju Kong. Multi-tenant data dynamic migration strategy of saas application in the cloud. *Computer Engineering and Science*, 35(10):89–97, 2013.

[10] Lanju Kong, Qingzhong Li, and Xiaona Li. A multi-tenant data migration policy for saas delivery platform. *Computer Applications and Software*, 28(11):52–56, 2011.

[11] Gerhard J Woeginger. Exact algorithms for np-hard problems: A survey. In *Combinatorial Optimization—Eureka, You Shrink!*, pages 185–207. Springer, 2003.

[12] Rania Hassan, Babak Cohanim, Olivier De Weck, and Gerhard Venter. A comparison of particle swarm optimization and the genetic algorithm. In *Proceedings of the 1st AIAA multidisciplinary design optimization specialist conference*, pages 1–13, 2005.

[13] Carlos A Souza Lima, Celso Marcelo F Lapa, Cláudio Márcio do NA Pereira, João J da Cunha, and Antonio Carlos M Alvim. Comparison of computational performance of ga and pso optimization techniques when designing similar systems–typical pwr core case. *Annals of Nuclear Energy*, 38(6):1339–1346, 2011.

[14] Ayed Salman, Imtiaz Ahmad, and Sabah Al-Madani. Particle swarm optimization for task assignment problem. *Microprocessors and Microsystems*, 26(8):363–371, 2002.

[15] Zhanghui Liu and Xiaoli Wang. A pso-based algorithm for load balancing in virtual machines of cloud computing environment. In *Advances in Swarm Intelligence*, pages 142–147. Springer, 2012.

[16] Ashkan Paya and Dan Marinescu. Energy-aware load balancing and application scaling for the cloud ecosystem. *IEEE Transactions on Cloud Computing*, 2015.

[17] Fahimeh Ramezani, Jie Lu, and Farookh Khadeer Hussain. Task-based system load balancing in cloud computing using particle swarm optimization. *International Journal of Parallel Programming*, 42(5):739–754, 2014.

[18] Andrew W Mcnabb, Christopher K Monson, and Kevin D Seppi. Mrpso: Mapreduce particle swarm optimization. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 177–177. ACM, 2007.

[19] Ibrahim Aljarah and Simone Ludwig. Parallel particle swarm optimization clustering algorithm based on mapreduce methodology. In *Nature and Biologically Inspired Computing (NaBIC), 2012 Fourth World Congress on*, pages 104–111. IEEE, 2012.

[20] Andrew W McNabb, Christopher K Monson, and Kevin D Seppi. Parallel pso using mapreduce. In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pages 7–14. IEEE, 2007.

[21] Leonardo Neumeyer, Bruce Robbins, Anish Nair, and Anand Kesari. S4: Distributed stream computing platform. In *Data Mining Workshops (ICDMW), 2010 IEEE International Conference on*, pages 170–177. IEEE, 2010.

[22] Ankit Toshniwal, Siddarth Taneja, Amit Shukla, Karthik Ramasamy, Jignesh M Patel, Sanjeev Kulkarni, Jason Jackson, Krishna Gade, Maosong Fu, Jake Donham, et al. Storm@ twitter. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 147–156. ACM, 2014.

[23] Kejiang Ye, Xiaohong Jiang, Dawei Huang, Jianhai Chen, and Bei Wang. Live migration of multiple virtual machines with resource reservation in cloud computing environments. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 267–274. IEEE, 2011.

[24] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[25] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the twenty-ninth annual ACM symposium on*

Theory of computing, pages 654–663. ACM, 1997.

[26] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: amazon's highly available key-value store. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 205–220. ACM, 2007.

[27] Haikun Liu, Hai Jin, Cheng-Zhong Xu, and Xiaofei Liao. Performance and energy modeling for live migration of virtual machines. *Cluster computing*, 16(2):249–264, 2013.

[28] Beth Trushkowsky, Peter Bodík, Armando Fox, Michael J Franklin, Michael I Jordan, and David A Patterson. The scads director: Scaling a distributed storage system under stringent performance requirements. In *FAST*, pages 163–176, 2011.

[29] Seyed Ebrahim Dashti and Amir Masoud Rahmani. Dynamic vms placement for energy efficiency by pso in cloud computing. *Journal of Experimental & Theoretical Artificial Intelligence*, (ahead-of-print):1–16, 2015.

[30] Zhen Xiao, Weijia Song, and Qi Chen. Dynamic resource allocation using virtual machines for cloud computing environment. *Parallel and Distributed Systems, IEEE Transactions on*, 24(6):1107–1117, 2013.

[31] Hector M Lugo-Cordero, Abigail Fuentes-Rivera, Ratan K Guha, Eduardo Ortiz-Rivera, et al. Particle swarm optimization for load balancing in green smart homes. In *Evolutionary Computation (CEC), 2011 IEEE Congress on*, pages 715–720. IEEE, 2011.

[32] Ibrahim Aljarah and Simone A Ludwig. Towards a scalable intrusion detection system based on parallel pso clustering using mapreduce. In *Proceedings of the 15th annual conference companion on Genetic and evolutionary computation*, pages 169–170. ACM, 2013.

[33] Simone A Ludwig. Mapreduce-based optimization of overlay networks using particle swarm optimization. In *Proceedings of the 2014 conference on Genetic and evolutionary computation*, pages 1031–1038. ACM, 2014.

[34] Jianhua Lin. Divergence measures based on the shannon entropy. *Information Theory, IEEE Transactions on*, 37(1):145–151, 1991.