International Journal of Contemporary Research in Computer Science and Technology (IJCRCST) e-ISSN: 2395-5325
Volume 2, Issue 1 (January '2016)

A REVIEW ON HEURISTIC TASK SCHEDULING FOR CLOUD COMPUTING

G.Keerthika,
M.Phil Research Scholar,
Department of Computer Science and Applications,
Vivekanandha College of Arts and Science College for Women (Autonomous),
Elayampalayam,Namakkal, Tamilnadu.

V.P.Muthukumar,
Assistant Professor,
Department of Computer Science and Applications,
Vivekanandha College of Arts and Science College for Women (Autonomous),
Elayampalayam,Namakkal, Tamilnadu.

Abstract: Cloud computing, the long-held dream of computing as a utility has the potential to transform a large part of the IT industry, making software even more attractive as a service and shaping the way in which hardware is designed and purchased. Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. The resource and scheduling allocation problem in a nature of NP-complete, which requiring the development of heuristic techniques to solve the resource allocation problem in a cloud computing environment. In this paper begins with a general introduction of cloud computing, followed by a discussion of heuristic task scheduling models.

Keywords: Cloud computing, heuristic, scheduling, resource allocation

I. INTRODUCTION

Since 2007, the term Cloud has become one of the most buzz words in IT industry. Lots of researchers try to define cloud computing from different application aspects, but there is not a consensus definition on it. I. Foster define as [1]: "A large-scale distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted virtualized, dynamically-scalable, managed computing power, storage, platforms, and services are delivered on demand to external customers over Internet." As an academic representative, Foster focuses on several technical features that differentiate cloud computing from other distributed computing paradigms. For example, computing entities are virtualized and delivered as services, and these services are dynamically driven by economies of scale.

a) Deployment models

Clouds are deployed in different fashions, depending on the usage scopes. There are four primary cloud deployment models [2].

Public cloud is the standard cloud computing paradigm, in which a service provider makes resources, such as applications and storage, available to the general public over Internet. Service providers charge on a fine-grained utility computing basis. Examples of public clouds include Amazon Elastic Compute Cloud (EC2), IBM's Blue Cloud, Sun Cloud, Google AppEngine and Windows Azure Services Platform.

Private cloud looks more like a marketing concept than the traditional main stream sense. It describes a proprietary computing architecture that provides services to a limited number of people on internal networks. Organizations needing accurate control over their data will prefer private cloud, so they can get all the scalability, metering, and agility benefits of a public cloud without ceding control, security, and recurring costs to a service provider. Both eBay and HP CloudStart yield private cloud deployments.

Hybrid cloud uses a combination of public cloud, private cloud and even local infrastructures, which is typical for most IT vendors. Hybrid strategy is proper placement of workloads depending upon cost and operational and compliance factors. Major vendors including HP, IBM, Oracle and VMware create appropriate plans to leverage a mixed environment, with the aim of delivering services to the business. Users can deploy an application hosted on a hybrid infrastructure, in which some nodes are running on real physical hardware and some are running on cloud server instances.

b) Cloud service

As an underlying delivery mechanism, cloud computing ability is provisioned as services, basically in three levels: software, platform and infrastructure [3].

Software as a Service: Software as a Service (SaaS) is a software delivery model in which applications are accessed by a simple interface such as a web browser over Internet. The users are not concerned with the underlying cloud infrastructure including network, servers, operating systems, storage, platform, etc. This model also eliminates the needs to install and run the application on the local computers. The term of SaaS is popularized by Salesforce.com, which distributes business software on a subscription basis, rather than on a traditional on-premise basis. Applications such as social media, office software, and online games enrich the family of SaaS-based services, for instance, web Mail, Google Docs, Microsoft online, NetSuit, MMOG Games, Facebook, etc.

Platform as a Service: Platform as a Service (PaaS) offers a high-level integrated environment to build, test, deploy and host customer-created or acquired applications. Generally, developers accept some restrictions on the type of software that can write in exchange for built-in application scalability. Customers of PaaS do not manage the underlying infrastructure

as SaaS users do, but control over the deployed applications and their hosting environment configurations.Typical examples of PaaS are Google App Engine, Windows Azure, Engine Yard, Force.com, Heroku, MTurk.

Infrastructure as a Service: Infrastructure as a Service (IaaS) provides processing, storage, networks, and other fundamental computing resources to users. IaaS users can deploy arbitrary application, software, operating systems on the infrastructure, which is capable of scaling up and down dynamically. IaaS user sends programs and related data, while the vendor's computer does the computation processing and returns the result. The infrastructure is virtualized, flexible, scalable and manageable to meet user requirements. Examples of IaaS include Amazon EC2, VPC, IBM Blue Cloud, Eucalyptus, FlexiScale, Joyent, Rackspace Cloud, etc.

II. SCHEDULING PROBLEMS

Scheduling problem [4] is the problem of matching elements from different sets, which is formally expressed as a triple (E, S, O), where,

- E is the set of examples, each of which is an instance of problem.
- S is the set of feasible solutions for the example.
- O is the object of the problem.

Scheduling problem can be further classified into two categories depending on object O: optimization problem and decision problem. An optimization problem requires finding the best solution among all the feasible solutions in set S. Different from optimization, the aim of decision problem is relatively easy. For a specified feasible solution s ∈ S, problem needs a positive or negative answer to whether the object O is achieved. Clearly, optimization problem is harder than decision problem, because the specified solution only compares with one threshold solution in decision problem, instead of all feasible solutions in optimization problem. An algorithm is a collection of simple instructions for finding a solution to a problem. It contains three parts: input, method, output. Input is a set of parameters to be dealt with. Method includes describable, controllable, repeatable procedures to realize the aim using input parameters. Output is a result of the problem. Especially for scheduling, the algorithm is a method by which tasks are given access, matched, or allocated to processors. Generally speaking, no absolutely perfect scheduling algorithm exists, because scheduling objectives may conflict with one another. A good scheduler implements a suitable compromise, or applies combination of scheduling algorithms according to different applications.

A problem can be solved in seconds, hours or even years depending on the algorithm applied. The efficiency of an algorithm is evaluated by the amount of time necessary to execute it. The running time of an algorithm is stated as a time complexity function relating the input length to the number of steps. There are several kinds of time complexity algorithms that will appear in the following,

- For a constant time algorithm O(1), the maximum amount of running time is bounded by a value that does not rely upon the size of the input.
- For a linear time algorithm O(n), the maximum amount of running time increases linearly with the size of the input.
- For a polynomial time algorithm O(nc) with a constant c, the maximum amount of running time is bounded by a polynomial expression in the size of the input.
- For a exponential time algorithm O(2nc) with a constant c, the maximum amount of running time is bounded by an exponential expression in the size of the input.

If a problem has a polynomial time algorithm, the problem is tractable, feasible, efficient or fast enough to be executed on a computational machine. In computational complexity theory, a complexity class is a set of problems that has the same complexity based on a certain resource [5].

- Class P is the set of decision problems that are solvable in polynomial time on a deterministic Turing machine, which means that a problem of Class P can be decided quickly by a polynomial time algorithm.
- Class NP is the set of decision problems that are solvable in polynomial time on a nondeterministic Turing machine, but a candidate solution of the problem of Class NP can be verified by a polynomial time algorithm, which means that the problem can be verified quickly.
- Class NP-complete is the set of decision problems, to which all other NP problems can be polynomial transformable, and a NP-complete problem must be in class NP. Generally speaking, NP-complete problems are more difficult than NP problems.
- Class NP-hard is the set of optimization problems, to which all NP problems can be polynomial transformable, but a NP-hard problem is not necessarily in class NP. Although most of NP-complete problems are computationally difficult, some of them are solved with acceptable efficiency. There are some algorithms, the running time of which is not only bounded by the size of input of an example, but also by the maximum number of the examples. These algorithms have pseudo polynomial time complexity. For one problem, if its maximum number is not large, it can be solved quickly. Thus, one NP-complete problem with known pseudo-polynomial time algorithms is called weakly NP-complete, otherwise is called strongly NP-complete, if it cannot be solved by a pseudo polynomial time algorithm unless P=NP [5].

III. HEURISTIC MODELS FOR TASK-EXECUTION SCHEDULING

Scheduling problems belong to a broad class of combinational optimization problems aiming at finding an optimal matching from a finite set of objects, so the set of feasible solutions is usually discrete rather than continuous. In cloud computing, a

typical datacenter consists of commodity machines connected by high speed links. This environment is well suited for the computation of large, diverse group of tasks. Tasks belonging to different users are no longer distinguished one from another. Scheduling problem in such a context turns out to be matching multi tasks to multi machines. As mentioned in the former section, the optimal matching is an optimization problem, generally with NP-complete complexity. Heuristic is often applied as a suboptimal algorithm to obtain relatively good solutions. This section intensively researches two types of strategies, static and dynamic heuristics. Static heuristic is suitable for the situation where the complete set of tasks is known prior to execution, while dynamic heuristic performs the scheduling when a task arrives. Before further explanation, several preliminary terms should be defined.

- t_i: task $_i$
- m_j : machine $_j$
- c_i: the time when task t_i comes
- a_j : the time when machine m_j is available
- e_{ij} : the execution time for t_i is executed on m_j
- c_{ij} : the time when the execution of t_i is finished on m_j , $c_{ij} = a_j + e_{ij}$
- makespan: the maximum value of c_{ij} , which means the whole execution time.

The aim of heuristics is to minimize makespan, that is to say, scheduling should finish execution of metatask as soon as possible.

a) Static strategies

Static strategies are performed under two assumptions. The first is that tasks arrive simultaneously $c_i = 0$. The second is that machine available time a_j is updated after each task is scheduled.

- ✓ **OLB** (Opportunistic Load Balancing) schedules every task, in arbitrary order, to next available machine. Its implementation is quite easy, because it does not need extra calculation. The goal of OLB is simply keeping all machines as busy as possible.

- ✓ **MET** (Minimum Execution Time) schedules every task, in arbitrary order, to the machine which has the minimum execution time for this task. MET is also very simple, giving the best machine to each task, but it ignores the availability of machines. MET jeopardizes the load balance across machines.

- ✓ **MCT** (Minimum Completion Time) schedules every task, in arbitrary order, to the machine which has the minimum completion time for this task. However, in this heuristic, not all tasks can be given the minimum execution time.

- ✓ **Min-min** begins with the set T of all unscheduled tasks. Then, the matrix for minimum completion time for each task in set T is calculated. Task with overall minimum completion time is scheduled to its corresponding machine. Next, the scheduled task is removed from T. The process repeats until all tasks are scheduled.

- ✓ **Min-max** is similar to Min-min heuristic. Min-max also begins with the set T of all unscheduled tasks, and then calculates the matrix for minimum completion time for each task in set T. Different from min-min, task with overall maximum completion time is selected and scheduled to its corresponding machine. Next, the scheduled task is removed from T. The process repeats until all tasks are scheduled.

- ✓ **GA** (Genetic Algorithm) is a heuristic to search for a near-optimal solution in large solution spaces [6]. The first step is randomly initializing a population of chromosomes (possible scheduling) for a given task. Each chromosome has a fitness value (makespan) that results from the scheduling of tasks to machines within that chromosome. After the generation of the initial population, all chromosomes in the population are evaluated based on their fitness value, with a smaller makespan being a better mapping. Selection scheme probabilistically duplicates some chromosomes and deletes others, where better mappings have a higher probability of being duplicated in the next generation. The population size is constant in all generations. Next, the crossover operation selects a random pair of chromosomes and chooses a random point in the first chromosome. Crossover exchanges machine assignments between corresponding tasks. Mutation operation is performed after crossover. Mutation randomly selects a chromosome, then randomly selects a task within the chromosome, and randomly reassigns it to a new machine. After evaluating the new population, another iteration of GA starts, including selection, crossover, mutation and evaluation. Only when stopping criteria are met, the iteration will stop.

- ✓ **SA** (Simulated Annealing) uses a procedure that probabilistically allows poorer solutions to be accepted to obtain a better search of the solution space. This probability is based on a system temperature that decreases for each iteration, which implies that a poorer solution is difficulty to be accepted. The initial system temperature is the makespan of the initial scheduling, which is mutated in the same manner as the GA. The new makespan is evaluated at the end of each iteration. A worse makespan might be accepted based on a probability, so the SA finds poorer solutions than Min-min and GA.

- ✓ **Tabu** search keeps track of the regions of the solution space which have already been searched so as not to repeat a search near these areas. A scheduling solution uses the same representation as a chromosome in the GA approach. To manipulate the current solution and to move through the solution space, a short hop is performed. The intuitive purpose of a short hop is to find the nearest local minimum solution within the solution space. When the short hop

procedure ends, the final scheduling from the local solution space search is added to the tabu list. Next, a new random scheduling is generated, to perform a long hop to enter a new unsearched region of the solution space. After each successful long hop, the short hop procedure is repeated. After the stopping criterion is satisfied, the best scheduling from the tabu list is the final answer.

A* is a tree-based search heuristic beginning at a root node that is a null solution. As the tree grows, nodes represent partial scheduling (a subset of tasks is assigned to machines), and leaves represent final scheduling (all tasks are assigned to machines). The partial solution of a child node has one more task scheduled than the parent node. Each parent node can be replaced by its children. To keep execution time of the heuristic tractable, there is a pruning process to limit the maximum number of active nodes in the tree at any one time. If the tree is not pruned, this method is equivalent to an exhaustive search. This process continues until a leaf (complete scheduling) is reached. The listed heuristics above are fit for different scheduling scenarios. The variation of scenarios is caused by the task heterogeneity, machine heterogeneity and machine inconsistence. The machines are consistent if machine mi executes any task faster than machine mj , it executes all tasks faster than mj . [6] For consistent machines, GA performs the best, while MET performs the worst. For inconsistent machines, GA and A* give the best solution, and OLB gives the worst. Generally, GA, A* and min-min can be used as a promising heuristic with short average makespan.

b) Dynamic strategies

Dynamic heuristics are necessary when task set or machine set is not fixed. For example, not all tasks arrive simultaneously, or some machines go offline at intervals. The dynamic heuristics can be used in two fashions, on-line mode and batch mode. In the former mode, a task is scheduled to a machine as soon as it arrives. In the latter mode, tasks are firstly collected into a set that is examined for scheduling at prescheduled times.

- ✓ **On-line mode** In on-line heuristics, each task is scheduled only once, the scheduling result can not be changed. On-line heuristic is suitable for the cases in which arrival rate is low [7].

- ✓ **OLB** dynamic heuristic assigns a task to the machine that becomes ready next regardless of the execution time of the task on that machine.

- ✓ **MET** dynamic heuristic assigns each task to the machine that performs that task's computation in the least amount of execution time regardless of machine available time.

- ✓ **MCT** dynamic heuristic assigns each task to the machine, which results in task's earliest completion time. MCT heuristic is used as a benchmark for the on-line mode [7].

- ✓ **SA** (Switching Algorithm) uses theMCT andMET heuristics in a cyclic fashion depending on the load distribution across the machines. MET can choose the best machine for tasks but might assign too many tasks to same machines, while MCT can balance the load, but might not assign tasks machines that have their minimum executing time. If the tasks are arriving in a random mix, it is possible to use the MET at the expense of load balance up to a given threshold and then use the MCT to smooth the load across the machines.

- ✓ **KPB** (K-Percent Best) heuristic considers only a subset of machines while scheduling a task. The subset is formed by picking the k best machines based on the execution times for the task. A good value of k schedules a task to a machine only within a subset formed from computationally superior machines. The purpose is to avoid putting the current task onto a machine which might be more suitable for some yet-to-arrive tasks, so it leads to a shorter makespan as compared to the MCT. For all the on-line mode heuristics, KPB outperforms others in most scenarios [7]. The results ofMCT are good, only slightly worse than KPB, owing to the lack of prediction for task heterogeneity.
- ✓ **Batch mode** In batch mode, tasks are scheduled only at some predefined moments. This enables batch heuristics to know about the actual execution times of a larger number of tasks.

- ✓ **Min-min** firstly updates the set of arrival tasks and the set of available machines, calculating the corresponding expected completion time for all ready tasks. Next, the task with the minimum earliest completion time is scheduled and then removed from the task set. Machine available time is updated, and the procedure continues until all tasks are scheduled.

- ✓ **Max-min** heuristic differs from the Min-min heuristic where the task with the maximum earliest completion time is determined and then assigned to the corresponding machine. The Max-min performs better than the Min-min heuristic if the number of shorter tasks is larger than that of longer tasks.

- ✓ **Sufferage** heuristic assigns a machine to a task that would suffer most if that particular machine was not assigned to it. In every scheduling event, a sufferage value is calculated, which is the difference between the first and the second earliest completion time. For task tk, if the best machine mj with the earliest completion time is available, tk is assigned to mj . Otherwise, the heuristic compares the sufferage value of tk and ti, the task already assigned to mj . If the sufferage value of tk is bigger, ti is unassigned and added back to the task set. Each task in set is considered only once.

Generally, Sufferage gives the smallest makespan among batch mode heuristics . The batch mode performs better than the on-line mode with high task arrival rate.

C) Heuristic schedulers

One advantage of cloud computing is that tasks which might be difficult, time consuming, or expensive for an individual user can be efficiently accomplished in datacenter. Datacenter in clouds supports functional separation between the processing power and data storage, both of which locate in large number of remote devices. Hence, scheduling becomes more complicated and challenging than ever before. Since scheduler is only a basic component for the whole infrastructure, no general scheduler can fit for all cloud architectures. In this section, we mainly discuss schedulers used for data-intensive distributed applications.

- ✓ **Hadoop** MapReduce is a popular computation framework for processing large-scaled data in mainstream public and private clouds, and it is considered as an indispensable cornerstone for cloud implementation. Hadoop is the most widespread MapReduce implementation for educational or production uses. It enables applications to work with thousands of nodes and petabytes of data. A multi-node Hadoop cluster contains two layers. The bottom is Hadoop Distributed File System (HDFS), which provides data location awareness for effective scheduling of work. Above the file systems is the MapReduce engine, which includes one job tracker and several task trackers. Every tracker inhabits an individual node. Clients submit MapReduce jobs to job tracker, then job tracker pushes work out to available Task Tracker nodes in the cluster [8]. Hadoop is designed for large batch jobs. The default scheduler uses FIFO heuristic to schedule jobs from a work queue. Alternative job schedulers are fair scheduler, capacity scheduler and delay scheduler.

- ✓ **FIFO scheduler** [8] applies first in first out heuristic. When a new job is submitted, scheduler puts it in the queue according to its arrival time. The earliest job on the waiting list is always executed first. The advantages are that the implementation is quite easy and that the overhead is minimal. However, throughput of FIFO scheduler is low, since tasks with long execution time can seize the machines.

- ✓ **Fair scheduler** [9] assigns equal share of resources to all jobs. When new jobs are submitted, tasks slots that free up are shared, so that each job gets roughly the same amount of CPU time. Fair scheduler supports job priorities as weights to determine the fraction of total compute time that each job should get. It also allows a cluster to be shared among a number of users. Each user is given a separate pool by default, so that everyone gets the same share of the cluster no matter how many jobs are submitted. Within each pool, fair sharing is used to share capacity between the running jobs. In addition, guaranteed minimum share is allowed. When a pool contains jobs, it gets at least its minimum share, but when the pool does not need its fullguaranteed share, the excess is split among other running jobs.
- ✓ **Capacity scheduler** [10] allocates cluster capacity to multiple queues, each of which contains a fraction of capacity. Each job is submitted to a queue, all jobs submitted to the same queue will have access to the capacity allocated to the queue. Queues enforce limits on the percentage of resources allocated to a user at any given time, so no user monopolizes the resource. Queues optionally support job priorities. Within a queue, jobs with high priority will have access to resources preferentially. However, once a job is running, it will not be preempted for a higher priority job.
- ✓ **Delay scheduler** [11] addresses conflict between scheduling fairness and data locality. It temporarily relaxes fairness to improve locality by asking jobs to wait for a scheduling opportunity on a node with local data. When the job that should be scheduled next according to fairness cannot launch a local task, it waits for a short length of time, letting other jobs launch tasks instead. However, if a job has been skipped long enough, it is allowed to launch non-local tasks to avoid starvation. Delay scheduler is effective if most tasks are short compared to jobs and if there are many slots per node.
- ✓ **Dryad** [12] is a distributed execution engine for general data parallel applications, and it seems to be Microsoft's programming framework, providing similar functionality as Hadoop. Dryad applies directed acyclic graph (DAG) to model applications.
- ✓ **Quincy** [13] scheduler tackles the conflict between locality and scheduling in Dryad framework. It represents the scheduling problem as an optimization problem. Min-cost flow makes a scheduling decision, matching tasks and nodes. The basic idea is killing some of the running tasks and then launching new tasks to place the cluster in the configuration returned by the flow solver.

Others

To sum up the heuristic schedulers for cloud computing, scheduling in clouds are all about resource allocation, rather than job delegation in HPC or grid computing. However, the traditional meta-schedulers can be evolved to adapt cloud architectures and implementations, considering the development of virtualization technologies. Next, we take several representatives for example as follows

- ✓ **Oracle Grid Engine** [14] is an open source batch-queuing system. It is responsible for scheduling remote execution of large numbers of standalone, parallel or interactive user jobs and managing the allocation of distributed resources. Now it is integrated by Hadoop and Amazon EC2, and works as a virtual machine scheduler for Nimbus in cloud computing environment.

- ✓ **Maui Cluster Scheduler** [15] is an open source job scheduler for clusters and supercomputers, which is capable of supporting an array of scheduling policies,

dynamic priorities, extensive reservations, and fair share capabilities. Now it has developed new features including virtual private clusters, basic trigger support, graphical administration tools, and a Web-based user portal in Moab.

- ✓ **Condor** [16] is an open source high-throughput computing software framework to manage workload on a dedicated cluster of computers. Condor-G is developed, provisioning virtual machines on EC2 through the VM Universe. It also supports launching Hadoop MapReduce jobs in Condor's parallel universe.
- ✓ **gLite** [17] is a middleware stack for grid computing initially used in scientific experiments. It provides a framework for building grid applications, tapping into the power of distributed computing and storage resources across the Internet, which can be compared to corresponding cloud services such as Amazon EC2 and S3. Since technologies such as REST, HTTP, hardware virtualization and BitTorrent displaced existing accesses to grid resources, gLite federates both resources from academic organizations as well as commercial providers to keep being pervasive and cost effective.

IV.CONCLUSION

In this paper, we firstly review the scheduling problems in a general fashion. Scheduling problem is to match multi tasks to multi machines, which can be solved by heuristics. Heuristics are classified into two types. Static heuristic is suitable for the situation where the complete set of tasks is known prior to execution, while dynamic heuristic performs the scheduling when tasks arrive. In cloud-related frameworks such as Hadoop and Dryad, batch-mode dynamic heuristics are most used, and more practical schedulers are developed for special usage. For commercial purpose, cloud services heavily emphasize time guarantee. The ability to satisfy timing constraints of such real-time applications plays a significant role in cloud environment.

V.REFERENCES

[1] I. Foster, Y. Zhao, I. Raicu, and S. Lu. Cloud computing and grid computing 360-degree compared. In Proceedings of Grid Computing Environments Workshop, pages 1–10, 2008.

[2] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid - enabling scalable virtual organizations. International Journal of Supercomputer Applications, 15:2001, 2001.

[3] J. Blazewicz. Scheduling in Computer and Manufacturing Systems. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1996.

[4] M. Sipser. Introduction to the Theory of Computation. International Thomson Publishing, 1st edition, 1996.

[5] T. D. Braun, H. J. Siegel, N. Beck, L. L. B¨ol¨oni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. Journal of Parallel and Distributed Computing, 61:810–837, June 2001.

[6] M. M. Shoukat, M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund. Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. Journal of Parallel and Distributed Computing, 59:107–131, 1999.

[7] D. Borthakur. The Hadoop Distributed File System: Architecture and Design. The Apache Software Foundation, 2007.

[8] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica. Improving mapreduce performance in heterogeneous environments. In R. Draves and R. van Renesse, editors, Proceedings of Symposium on Operating Systems Design and Implementation, pages 29–42. USENIX Association, 2008.

[9] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. Job scheduling for multi-user mapreduce clusters. Technical Report UCB/EECS-2009-55, EECS Department, University of California, Berkeley, Apr 2009.

[10] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In Proceedings of International Conference on EuroSys, pages 265–278, 2010.

[11] Dryad. http://research.microsoft.com/en-us/projects/dryad/.

[12] M. Isard, V. Prabhakaran, J. Currey, U.Wieder, K. Talwar, and A. Goldberg. Quincy: fair scheduling for distributed computing clusters. In Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles, pages 261–276. ACM, 2009.

[13] Oracle grid engine. http://www.sun.com/software/sge/.

[14] Maui cluster scheduler. http://www.cluster.com/maui-cluster-scheduler.php.

[15] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: the condor experience: Research articles. Journal of Concurrency: Practice and Experience, 17:323–356, February 2005.

[16] C. Ragusa, F. Longo, and A. Puliafito. Experiencing with the cloud over glite. In Proceedings of ICSE Workshop on Software Engineering Challenges of Cloud Computing, pages 53–60. IEEE Computer Society, 2009.