Self-adaptive, Deadline-aware Resource Control in Cloud Computing

Yu Xiang[1], Bharath Balasubramanian[2], Michael Wang[2], Tian Lan[1], Soumya Sen[3], and Mung Chiang[2]

[1]Department of Electrical and Computer Engineering, George Washington University, DC 20052
{xy336699, tlan}@gwu.edu
[2]Department of Electrical and Engineering, Princeton University, NJ 08543
{bharathb, mwseven, chiangm}@princeton.edu
[3]Carlson School of Management, University of Minnesota, MN 55455
ssen@umn.edu

Abstract. Modern data centers deliver resources over the cloud for clients to run various applications and jobs with diverse requirements. Today's cloud resource management is able to support certain Quality of Service (QoS) requirements including reliability and security. However, in many settings such as the military cloud where latency requirement is paramount, existing cloud resource management schemes fall short in providing a systematic framework to meet and balance disparate types of application deadlines, since they are primarily focused on speeding up job executions for timely processing. In this paper we present a self-adaptive, deadline-aware resource control framework that can be implemented in a fully distributed fashion, making it suitable for unreliable environments where a single point of failure is not acceptable. Relying on Nash Bargaining in non-cooperative game theory, our framework allocates cloud resources in an optimal way to maximize the Nash Bargaining Solutions (NBS) with respect to both job priority and deadline. Further, it also enables self-adaptive deadline-aware resource allocation and rebalancing under cyber or physical attacks that may diminish cloud capacity. We validate our technique by performing experiments on the Hadoop framework.

I. INTRODUCTION

Cloud computing is becoming increasingly prevalent as it allows the delivery, in a pay-per-use manner, of highly automated, streamlined cloud services over a networked environment to hundreds of thousands of cloud clients with low costs and elastic Service Level Agreements (SLAs). As more applications are migrating toward the cloud, it poses a great challenge to meet disparate client expectations that vary significantly due to their heterogeneous requirements and demands, e.g., application type, priority and deadline, resource requirements, and personal budgets. All of these necessitate a highly flexible, adaptive, and resilient cloud resource management framework to support such requirements and dynamically adjust resource allocation.

In this paper, we focus on military clouds that need to provide self-adaptive, resilient cloud services in hostile forward areas and enable a "scattered cloud" to support missions with disparate demands. We dynamically adjust resource allocation and mission scheduling to allow the cloud to fight external attacks autonomously. In such a cloud system it is critical to know when, where, and how to execute various missions so that effectiveness of all tasks executing on the cloud is optimized. Hence this challenge needs novel solutions that offer self-adaptive, deadline-aware mission scheduling and resource allocation. We make use of Nash Bargaining [5] in non-cooperative game theory, which provides a single value solution that uniquely satisfies four axioms, i.e., Invariance to Affine Transformations, Pareto Optimality, Symmetry, and Independence of Irrelevant Alternatives. By extend Nash Bargaining to take into account disparate job priorities and deadlines, our proposed cloud framework allocates and dynamically rebalances cloud resources in an optimal way to maximize prioritized and deadline-aware Nash Bargaining Solutions (NBS) with respect to cloud capacity changes as well as job arrivals/departures.

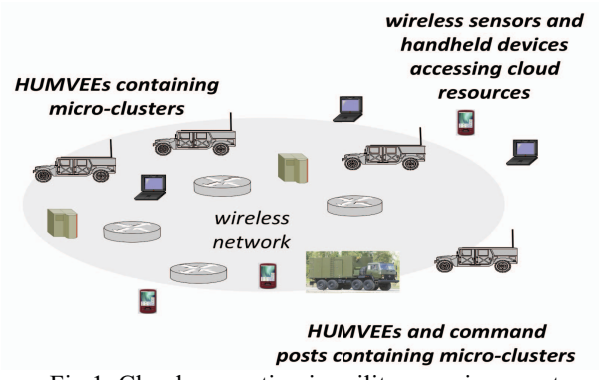

Fig 1. Cloud computing in military environments

978-1-4799-5086-7/13 $31.00 © 2013 IEEE
DOI 10.1109/SASOW.2013.35

The military cloud we model supports an architecture in which computing resources (both computing and storage capacity) reside in geographically dispersed micro-clusters, each of which contains only a fraction of overall cloud capacity. Micro-clusters might comprise a small rack of servers in the back of a HUMVEE, command post, aircraft, or other structure of opportunity, as shown in Fig. 1. The NBS framework developed in this paper can be implemented independently within each micro cluster to offer intra-cluster resource bargaining, while its output, including expected completion time and congestion price, provide valuable information to guide upper layers in the could system for inter-cluster resource management. We refer readers to [1, 2] for a survey of such cluster selection and load-balancing algorithms.

The rest of the paper is organized as follows. Following a review of prior relevant work in Section II, a background of Nash Bargaining and the problem formulation is introduced in Section III. We describe our proposed optimization algorithm in Section IV, and present our experimental results on the Hadoop framework [10] in Section V. Section VI summarizes this work and outlines several possible directions for future work.

II. Related Work

While existing approaches on cloud resource management have focused on improving timely execution of critical cloud applications via admission control, concurrent job execution, or preemption of lower priority jobs [1,3,4,6,7,8], relatively little past work has considered a systemic resource control framework to meet and dynamically rebalance disparate types of application deadlines in order to maximize overall mission effectiveness.

For Hadoop-based data processing, the authors in [1] propose a job execution cost model that accounts for the various parameters that affect job completion times and designed a constraint-based scheduler that takes user deadlines as part of its input and determines the feasibility of scheduling a job based on the proposed cost model. However, this solution does not allow any deadline optimization and jobs are only scheduled if specified deadlines can be met. Another related work [4] presents delay scheduling in Hadoop, aiming at solving the conflict between fair scheduling of jobs and latency for obtaining non-local data. In their approach, when the next job scheduled to run cannot be launched immediately due to data availability and locality, it gives up the scheduling opportunity to other jobs to improve overall execution efficiency despite some fairness degradation. While this approach speeds up job processing, the approach does not optimize resource allocation to meet individual deadline requirements. The goal of this paper is to develop a self-adaptive and deadline-aware resource control framework, which not only guarantee individual job deadlines when feasible, but also maximize overall cloud efficiency by adjusting and rebalancing mission objectives.

III. Nash Bargaining

In this section, we present the application of the Nash bargaining solution to our problem. Nash Bargaining introduces a form of utility maximization that yields efficient, Pareto-optimal allocation of resources [5], among clients in a prioritized and deadline-aware fashion. Our previous work [1] showed that this approach succeeds in balancing computing loads among dispersed micro-clusters (hereafter referred to simply as clusters) without the need for centralized cloud-wide control. To do so, cloud clients obtain estimates of computational congestion price (produced by the Nash Bargaining algorithm) from reachable clusters, and send their jobs to the cluster with the lowest price.

In this paper, we consider a three-step process for clients/cluster controller to allocate jobs across various clusters. First, a cluster controller submits the client jobs to all the clusters to obtain a congestion price. Second, each cluster calculates the congestion price based on the NBS solution. Finally, the coordinator uses this information to submit the job to the appropriate. In this section, we focus on the second step: calculating the congestion price based on the jobs in the system within each cluster.

Consider a set of N jobs, each job indexed by *i,* submitted to a cluster with C resources. For each job *i*, we associate a total resource requirement D_i, priority P_i and a utility function $U_i(y_i)$ that quantifies the value of the job to the client as a non-decreasing function of the aggregate units of resource y_i that it receives. The goal of Nash Bargaining is to maximize the total utility that all *N* jobs in the system can receive before their designated deadlines, weighted by their priorities. In addition, if the available resources change due to attack or failure, the system must rapidly and autonomously find new operating points that again attain the maximum aggregate utility achievable from the available resources. Nash Bargaining provides a solution to this problem. Specifically, it maximizes the generalized Nash product

$$\max \prod_i \left[U_i(y_i)\right]^{P_i} \text{ s.t., } y_i \text{ are feasible.} \quad (1)$$

Where the aggregate resource y_i must be feasible under cluster capacity constraints. The NBS approach provides a unique optimal solution to (1) with several attractive properties. It is efficient (i.e., it utilizes all available resources), achieves proportional fairness, and is also

Pareto optimal: it is not possible to increase any y_i without decreasing one or more other jobs' resource allocation. The NBS approach also achieves proportional fairness. By taking a logarithmic transformation, (1) becomes:

$$\max \sum_i P_i \cdot \log\left(U_i(y_i)\right) \text{ s.t., } y_i \text{ are feasible.} \quad (2)$$

Here NBS maximizes the sum of individual logarithmic utilities, weighted by their respective priorities. The significance of this point will become apparent shortly.

As discussed earlier, clients may have different requirement on job completion time, and in this paper we consider three different types of deadlines: (i) Hard deadlines, which cannot be violated, and a job must be assigned enough resource to complete by the deadline, (ii) Soft deadlines, which can be exceeded but any resource allocated after the deadline will have decreasing utility value, and (iii) No deadlines, which means the job does not have to meet any deadline at all, and resources allocated at any time are equally useful. In a military cloud, examples of hard deadline jobs include missile trajectory calculation or maneuvering unmanned vehicles, while automated translation of recorded speech could have soft deadlines, and identifying enemy combatants from security/surveillance feeds may be an example of no deadline jobs. In the next, we mathematically define these types of deadlines within our NBS framework.

Hard deadline. Let $x_i(t)$ be the resource allocated to job *i* in time slot *t*. In the hard deadline case, the utility of a job is solely determined by the aggregate resources it receives from job submission to the deadline, i.e.,

$$y_i = \frac{1}{D_i} \int_0^{t_i^{dead}} x_i(t)dt \geq 1 \quad (3)$$

Here resource y_i is normalized by total resource requirement D_i to complete the job. For hard deadline jobs, we must have $y_i \geq 1$since each job *i* should receive at least D_i units of resource before the hard deadline.

Soft deadline. For this case, we compute aggregate resource allocation with respect to a time-dependent weigh function $w_i(t)$:

$$y_i = \frac{1}{D_i}\int_0^{t_i^{dead}} x_i(t)dt + \frac{1}{D_i}\int_{t_i^{dead}}^{t_i^{ref}} x_i(t)w_i(t)dt \quad (4)$$

$$w_i(t) = \begin{cases} 1, & \text{if } t \leq t_i^{dead} \\ e^{-\mu(t-t_i^{dead})}, & \text{if } t > t_i^{dead} \end{cases} \quad (5)$$

Since any resource assigned after a soft deadline is still useful, but would have diminishing utility, we weight resource $x_i(t)$ by a discount function $w_i(t)$ that is 1 before the deadline and decreases exponentially over time afterwards. Here t_i^{ref} represents the time horizon of NBS optimization. Decaying factor μ may vary based on application type and client expectations. There is no minimum resource requirement for soft deadline jobs.

No deadline. By our definition, resources assigned to no deadline jobs, at any time, always have equal utility value. Therefore, we obtain

$$y_i = \frac{1}{D_i}\int_0^{t_i^{ref}} x_i(t)dt \quad (6)$$

Here the aggregate resource is computed from job submission to the time horizon t_i^{ref}. Notice that, similar to soft deadline jobs, no minimum resource assignment is required for no deadline jobs.

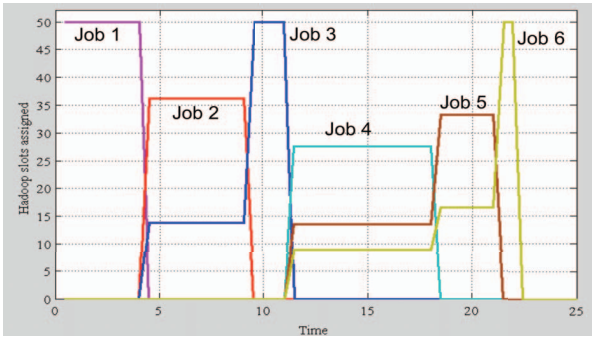

Fig 2. An example of NBS optimization.

These 3 types of deadlines are closely related. It is easy to see that a soft deadline job with $\mu = 0$ is equivalent to a no deadline job, while it corresponds to a hard deadline job for $\mu = +\infty$.

Combining the above equations for the various type of deadlines and taking into account the cluster capacity, we formulate the following self-adaptive, NBS optimization problem:

$$\max \sum_i P_i \cdot \log\left(U_i(y_i)\right),$$

$$\text{s.t., } \sum_i x_i(t) \leq C \ \forall t, \text{ and (3),(4),(5),(6).}$$

The problem described above is an extension of the classic optimization problem relying on NBS because it takes into account the relative, temporal value of resource allocation with respect to when it is assigned, in addition to "what" and "how much" is assigned. When utility function $U_i(y_i)$ satisfies certain conditions (e.g., linear

or logarithmic functions), it is easy to verify that the NBS optimization is a convex problem and thus can be solved by off-the-shelf optimization tools. The NBS optimization is performed repeatedly at each job arrival and departure, or when cloud capacity changes. Its solution also provides an estimated completion time and a resource congestion price, which guide upper layer modules for cluster selection and workload balancing.

Figure 2 illustrates a simple simulation of our algorithm for a cluster with 6 jobs. While all jobs have equal resource requirements, jobs 1 and 2 have hard deadlines and are guaranteed to completed before their deadlines at *t=5* and *t=10* (if feasible). Soft deadline jobs 3 and 4 completes close to their deadlines at *t=10* and *t=15*, but before any no deadline jobs 5 and 6 finish.

IV. IMPLEMENTATION AND RESULTS

In this section, we present experimental results for our NBS scheduling algorithm on the ubiquitously used Hadoop framework [10] for processing MapReduce [11] jobs. We compare our algorithm with three other scheduling policies: (i) Random scheduling in which each job is allocated a random share of the cluster resources, (ii) Fair or Equal scheduling in which each job is allocated the same share of cluster resources, (iii) Earliest-Deadline-First (EDF) scheduling in which jobs are processed in increasing order of their deadlines.

Our experiments focus on the performance of our algorithm w.r.t the latency experienced by the hard-deadline jobs and the latency experienced by high priority jobs. The results confirm the following facts: (a) For a set of jobs submitted to the Hadoop framework unlike the other three schedulers, our policy always finishes hard-deadline jobs before their deadline. (b) For a given set of job, our scheduler finishes the high priority jobs much before the deadline as compared to other schedulers. (c) For experiments averaged over many sets of jobs, the cumulative latency for hard-deadline jobs and high priority jobs is lesser than that for the other schedulers.

Hadoop is one of the most commonly used engines for large-scale data processing applications modeled as MapReduce jobs. The MapReduce framework is built using the master-worker configuration where the master assigns the map and reduce tasks to various workers. While the map tasks perform the actual computation on the data files received by it as <key, value> pairs, the reducer tasks aggregate the results according to the keys and writes it to the output file. Let us consider the canonical example of Hadoop MapReduce for the word count application in which we we wish to count the number of occurrences of each word in a given data set. For the word count job, the Hadoop framework first splits the data into smaller chunks and assigns map tasks for each of these chunks. The map tasks simply output <key, value> pairs corresponding to its input chunk, where the key is the string which occurs in the input text and the value is always 1. Then reduce tasks combine the outputs of each of these map tasks, which in this case is simple addition of the values corresponding to each key. This would give us the word count in the text.

Scheduling in Hadoop is primarily based on the number of map and reduces tasks the engine can allocate to each job in the system. For the purposes of our experiment we focus purely on the number of map tasks assigned to different jobs and consider a job completed when all its map tasks have executed.

We perform our experiments on a 3 machine, 12-core Hadoop cluster with a total capacity of 6 map tasks. All the jobs in our experiments are identical word count jobs each of which requires 3 map tasks to complete. We implemented a Hadoop-plugin that controls the number of map slots allocated to each job in the system, thereby controlling the resources allocated to them. Using this framework, we can specify different scheduling policies. We ran different sets of experiments with the four scheduling policies defined above. The results of our experiments are shown in Fig. 3, Fig. 4, Fig. 5 and Fig. 6.

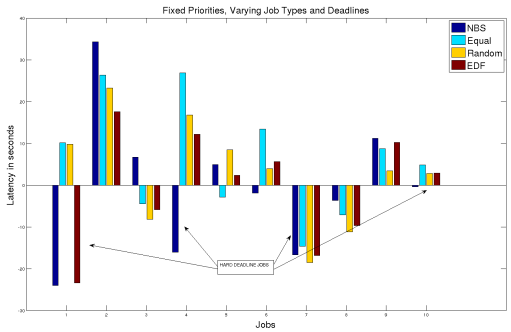

Fig 3. Latency distribution for Hard-Deadline Jobs

The results in Fig. 3 and Fig. 4 correspond to experiments where we submitted a set of ten jobs and collected the time taken for each of the jobs to complete. We ensured that among the jobs at least one-third had hard deadlines, while the remaining two-third jobs were assigned soft and no deadlines randomly. The priorities of all the jobs were the same and equal to 1 while the deadlines were fully randomized.

The graphs show the latency (execution time - deadline) experienced under each scheduling policy. The results confirm that our scheduling policy (NBS) always complete all the jobs with hard deadlines well before their deadline (when feasible). In fact while NBS has a worst-case latency of -0.367 secs, the 'Equal' (or fair), 'Rand'

and 'EDF' scheduling policies have worst-case latency of 26.84, 16.77 and 12.177 secs for hard deadline jobs respectively. In Fig. 4, we assign all jobs an identical deadline of 65 secs and job type of 1 (soft deadlines), while varying the priority from 300 to 30. Note that for jobs with priority greater than or equal to 100 (first three jobs on the graph starting from 0), the worst case latency by NBS is just -6.85 secs, whereas the latency for Equal, Rand and EDF are 8.45 secs, 13.77 secs and 6.128 secs respectively.

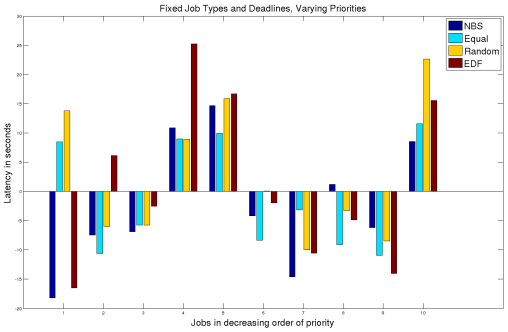

Fig 4. Latency distribution for High-Priority Jobs

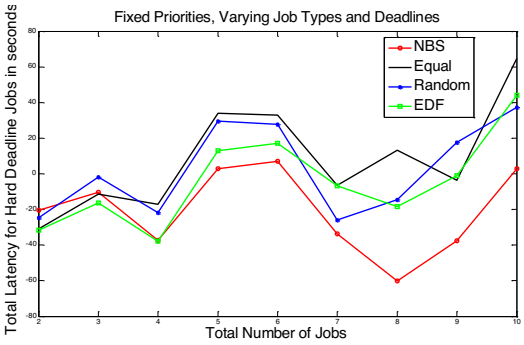

Fig 5. Aggregate Latency for Hard-Deadline Jobs

In Fig. 5 and Fig. 6, we show results that compare the aggregate performance of our scheduler across runs with different sets of jobs, where the number of jobs is varied from 2 to 10. In the first experiment (Fig. 5), we plot the latency for hard-deadline jobs. For each run, at least one-third of the jobs had hard deadlines, while the remaining two-third jobs were assigned soft and no deadlines randomly. The deadlines were varied randomly while the priorities were kept constant at 1. The results show that NBS on average, across the runs, has a latency of -20.72 secs for hard-deadline jobs, while Equal, Rand and EDF have average latencies of 2.665 secs, 8.405 secs and -4.2 secs respectively.

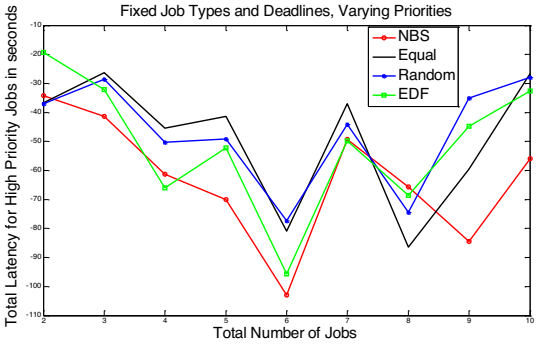

Fig 6. Aggregate Latency for High-Priority Jobs

In the second experiment (Fig. 6) we assign a constant job type of 1 and deadline of 65 seconds. The job priorities in each run was varied randomly 2 to 256. The total latency for jobs with priority greater than 50 was plotted. The results show that NBS on average, across the runs, has a latency of -62.8 secs for high priority jobs, while Equal, Rand and EDF have average latencies of -47.1 secs, -48.9 secs and -51.2 secs respectively. In conclusion, our experiments on Hadoop confirm that our scheduler prioritizes the scheduling of jobs with hard-deadlines and high priorities over other commonly used schedulers.

V. Conclusions

This paper proposes a self-adaptive, deadline-aware cloud resource control framework based on the Nash Bargaining Solution. Our technique apportions resources in an optimal way to maximize aggregate utility with respect to both job priorities and deadlines. It enables automated resource scheduling and rebalancing under cyber or physical attacks that may diminish cloud capacity. Our prototype implementation utilizing Hadoop Mapreduce shows significant utility improvement and deadline satisfaction over existing approaches.

References

[1] S. Wagner *et al*., "Autonomous, Collaborative Control for Resilient Cyber Defense," *in Proceedings of 2012 IEEE SASO*, Lyon, September 2012.

[2] S. Wagner *et al*., "Adaptive, Network-Aware Cluster Selection for Cloud Computing in Wireless Networks," *Submitted to 2013 IEEE SASO*, Philadelphia, July 2013.

[3] K. Anyanwu, "Scheduling Hadoop Jobs to Meet Deadlines," *in Proceedings of Cloud Computing, Second International Conference, CloudCom 2010*.

[4] M. Zaharia *et al*., "Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling", *In Proceedings of EuroSys,* 2010.

[5] J. Nash, "The Bargaining Problem", *Econometrica*, Vol. 18, No. 2, April 1950, pp. 155-162.

[6] T. Sandholm and K. Lai, "MapReduce optimization using regulated dynamic prioritization," *in Proceedings of the 11th International Joint Conference on Measurement and Modeling of Computer Systems* (SIGMETRICS'09), Seattle, USA, Jun. 2009, pp. 299–310.

[7] J. Polo, D. Carrera, Y. Bacerra, V. Beltran, J. Torres, and E. Ayguade, "Performance management of accelerated MapReduce workloads in heterogeneous clusters," *in Proceedings of the 39th International Con- ference on Parallel Processing (ICPP'10)*, San Diego, USA, Sep. 2010, pp. 653–662.
[8] M. Mattess *et al*., "Scaling MapReduce Applications across Hybrid Clouds to Meet Soft Deadlines", *in Proceedings of IEEE 27th International Conference on Advanced Information Networking and Applications*, 2013.
[9] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. Commun. ACM, 51:107–113, January 2008.
[10] T. White. Hadoop: The Definitive Guide. O'Reilly Media, Inc., 1st edition, 2009.
[11] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.