Autonomous, Collaborative Control for Resilient Cyber Defense (ACCORD)

Stuart Wagner, Eric van den Berg, Jim Giacopelli, Andrei Ghetie, Jim Burns, Miriam Tauil
Applied Communication Sciences, Inc.

Soumya Sen, Michael Wang, Mung Chiang
Princeton University

Tian Lan
George Washington University

Robert Laddaga, Paul Robertson, Prakash Manghwani
DOLL, Inc.

Abstract. ACCORD addresses the need for robust, rapidly adaptive resource allocation mechanisms in cloud computing. It employs a distributed, game-theoretic approach to apportion computational loads in an efficient, prioritized, Pareto-optimal fashion among geographically dispersed cloud computing infrastructure. This paper describes ACCORD algorithms, software implementation, and initial experimental results. Our results illustrate how a distributed, ACCORD-enabled cloud architecture autonomously adapts to the loss of computing resources (e.g., due to failures, poor network connectivity, or cyber attack) while ensuring that users receive maximal, prioritized utility from available cloud resources.

I. Introduction and Background

Traditional approaches to cloud computing [1] typically assume (1) a cloud implementation with one or a small number of large datacenters maintaining substantial computing or storage capacities; (2) centralized control of computing resources within or across datacenters; and (3) highly reliable network connectivity to datacenters. Although many commercial cloud computing scenarios adhere well to these assumptions, several critical emerging applications do not. Cloud computing in tactical military environments represents the best counter-example: deployment of monolithic datacenters in hostile forward areas is impractical and unreliable, because a single focused attack targeting the datacenter could disable all computing capacity. The same consideration argues against centralized control, because the controlling entity constitutes a single point of failure. In addition, tactical wireless networks are notoriously unreliable and bandwidth-constrained, and robust network access to any one physical location cannot be guaranteed. The potential for attacks against these networks, including jamming (both inadvertent and malicious), distributed denial of service, or cyber attack against routing infrastructure, will exacerbate these problems. Further heightening these concerns are recent studies indicating that the network can easily become the main cloud performance bottleneck even when it is not under attack [1]. Various subsets of these design challenges exist in other, non-military cloud usage scenarios such as emergency disaster response units, and research missions in remote areas.

ACCORD addresses these unique challenges in several ways. We support an architecture in which computing resources (both computing and storage capacity) reside in *geographically dispersed microclusters*, each of which contains only a fraction of overall cloud capacity. Microclusters 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 ability to support physically dispersed cloud resources has two critical advantages in tactical military and other similarly challenged environments. First, the distributed nature of the infrastructure makes it more difficult for adversaries to target large portions of those resources (e.g., via the types of attacks outlined above). Second, the physical distribution makes it more likely that a given client of the cloud will have at least some cloud resources in relatively close proximity, thereby improving the quality of network connectivity as measured by latency, throughput and connection persistence.

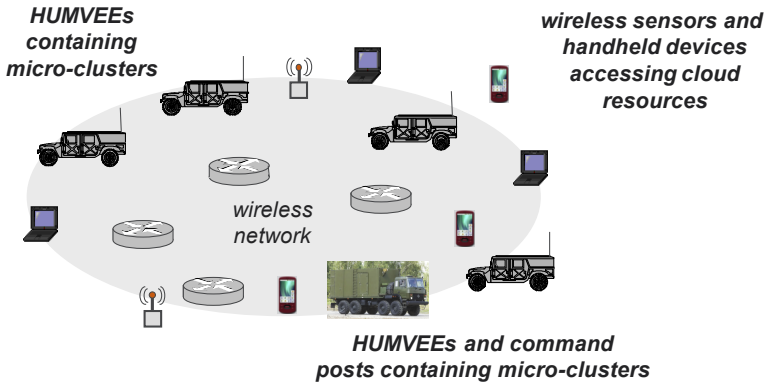

Fig. 1. ACCORD supports operation of geographically distributed clouds with mobile, wireless clients.

Although each microcluster contains a mechanism to control and allocate resources within that microcluster, cloud operation is otherwise fully distributed. ACCORD relies on a distributed, game-theoretic algorithm and protocol for efficient prioritization and load balancing across all microclusters reachable by clients of the cloud, as we explain in Section II. In Section III we describe an initial software implementation of the ACCORD architecture. We highlight initial experimental results in Section IV, showing how ACCORD's distributed, collaborative approach to resource allocation supports adaptive, resilient access to cloud services in the face of microcluster loss due to causes such as power-down, physical attack, cyber attack, or loss of network connectivity. Section V highlights some areas of ongoing investigation in the ACCORD project.

II. Distributed, Game-Theoretic Allocation of Computing Resources

Fundamental to our approach is the concept of Nash Bargaining [2,3], a form of cooperative game theory. In response to attacks or failures that diminish cloud capacity, ACCORD utilizes distributed bargaining algorithms to re-apportion available computational resources among users in a rapid, prioritized fashion. In general, an Asymmetric Nash Bargaining Solution (NBS) among N users will maximize the Nash product

$$\max \prod_{i=1}^{N} (U_i - F_i)^{p_i} \qquad (1)$$

where U_i is the user utility, p_i is the user priority, and F_i are the "disagreement points" that correspond to the utility that each user receives in the case of a breakdown in bargaining. For the purpose of this paper, we assume that $F_i = 0$; that is, users only derive utility in the case of

successful bargaining. The NBS provides a unique maximum with several attractive properties: it is efficient, achieves proportional fairness [4], and is also Pareto-Optimal: it is not possible to increase any U_i without decreasing one or more other U_i. The solution to the generalized Nash product (1) is the same as that for

$$\max \sum_{i=1}^{N} p_i \log(U_i) \qquad (2)$$

The NBS maximizes the sum of individual logarithmic utilities, weighted by their respective priorities.

Nash Bargaining apportions a cluster's computational resources among clients utilizing that cluster, and provides clients with utility and congestion price estimates to help them select the most favorable cluster(s) for future job execution. This selection may include intentional replication of jobs among multiple clusters to increase the probability of successful job completion. We explain the rationale and details of these steps in greater detail below.

Estimation of a cluster's processing delay goes hand-in-hand with the problem of optimal intra-cluster resource sharing. Jobs require resources in different relative quantities. Say job j uses a vector $\boldsymbol{Z}_j = \left(z_{1j}, z_{2j}, \dots, z_{mj}\right)^T$ of resources per scheduling time unit, e.g. z_{1j} MIPS of CPU, z_{2j} GB of memory, and z_{3j} Mb/s of I/O capacity. Given $\boldsymbol{Z}_j$ and priority level p_j, $j = 1, \dots, L$, the cluster controller can derive an NBS for Pareto-optimal mapping of resources to jobs. Let x_j be the effective processing rate achieved by job j in a given allocation period t, i.e., the number of times that job j can be completed within time slot t. Since jobs are by assumption elastic, if we allocate more resources to a single job, it will simply finish (proportionally) faster, but jobs are fungible in that they can operate with a fractional multiple of their profiles. This models the situation where a job consists of a large number of sub-tasks, each of which can be completed independently of the others, in parallel or sequentially. For L total jobs, the NBS for intra-cluster resource allocation solves the problem

$$\max \textstyle\sum_{j=1}^{L} p_j \log\left(x_j\right) \qquad subject\ to\ \boldsymbol{Z}\boldsymbol{x} \leq \boldsymbol{v} \qquad (3)$$

Here $\boldsymbol{x} = (x_1, x_2, \dots, x_L)^T$ is a column vector of the aforementioned processing rates for each job, $\boldsymbol{Z} = [\boldsymbol{Z}_1, \dots, \boldsymbol{Z}_L]$ is the matrix of resource usage profiles for each job, and $\boldsymbol{v} = (v_1, v_2, \dots, v_m)^T$ is a column vector of the resource bounds in each dimension. The Lagrangian of (3) is

$$L(x, \lambda, \mu) = \textstyle\sum_{j=1}^{L}\left\{p_j \log\left(x_j\right) - \left(\sum_{r=1}^{m} \lambda_r z_{rj}\right)x_j\right\} + \sum_{r=1}^{m} \lambda_r v_r \qquad (4)$$

For each job j, we can define a per-unit-rate congestion price $q_j = \sum_{r=1}^{m} \lambda_r z_{rj}$. At optimality, we have $1/x_j^* = q_j^*/p_j = \sum_{r=1}^{m} \lambda_r^* z_{rj}/p_j$. The cluster controller conveys the optimal q_j to the client submitting the job request, which (because the client knows p_j) enables the client to estimate both expected utility and resource cost in selecting this cluster for the job.

III. Implementation of NBS and Client-Cluster Protocol

To achieve distributed allocation of computing resources across the cloud, ACCORD utilizes software residing at clients and at microclusters (hereafter referred to simply as clusters), along with a client-cluster control protocol. Figure 2 illustrates this architecture at a high level, for the case where a client has access to three separate clusters. Each cluster contains a cluster controller that accepts job pricing queries from authorized, authenticated clients and returns a "congestion

price" which reflects to the aggregate load within that cluster. The job pricing requests consist of a description of the job to be executed and the desired priority level. The congestion price is computed based on the cumulative number of jobs currently being executed within the cluster, weighted by the priority of each job. The congestion price is proportional to the computing load currently within the cluster. As the cluster load increases, the cost reflected by the congestion price also increases.

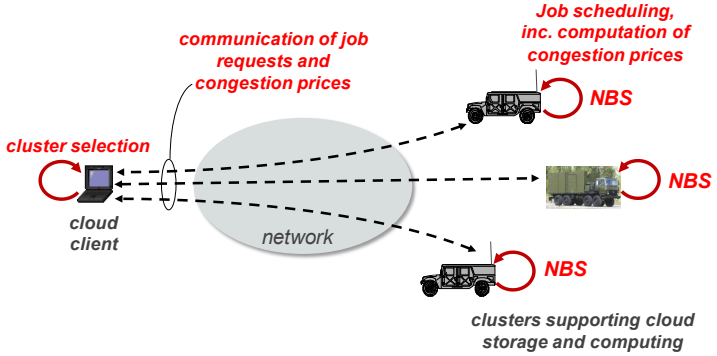

Fig. 2. Architecture and protocol for cluster selection by clients of the cloud.

Clients independently make cluster selections based on the congestion price received via the protocol described below. The client will typically submit its job to the cluster returning the best price. Upon receipt of a job execution request, the cluster controller will formulate the job and submit the job request to Hadoop for execution. The cluster controller maintains status of the jobs in progress and revises its congestion pricing to reflect the new job, if and when future price queries are received.

Apache™ Hadoop™ (hadoop.apache.org) from The Apache Software Foundation was used as the cloud environment to demonstrate the principles of a distributed NBS solution. A Hadoop cluster consists of a series of tightly coupled relatively low cost Linux nodes. Storage is distributed among all machines within the cluster and replicated for reliability using the Hadoop distributed file system. Hadoop through its map-reduce framework provides for rapid elasticity in the allocation of computing resources to jobs. Jobs within Hadoop are broken down into independent tasks that can be completed in parallel or sequentially, depending on available resources and priorities of other jobs currently being executed within the cluster.

evdb 6/7/12 5:25 PM
Comment [1]: Added url for Hadoop

The rate at which tasks from a given job are executed is dependent on the priority of the tasks comprising a job and the level of congestion within the Hadoop cluster. Control of task scheduling is done by the Hadoop fair scheduler. The fair scheduler enables rates to be assigned to jobs and proportionally schedules tasks for execution based on these assigned rates. The rates are a relative weighing of job pools within Hadoop where each pool services a given priority level. Pools allow concurrent execution of jobs within the cluster. Our cluster controller adjusts the respective weights assigned to each pool which in turn manages the allocation of computing resources to each job. As more jobs at a given priority level are offered to the system, the proportion of resources allocated to the pool is increased, thus providing a higher level of computing resources to higher priority jobs and reducing the allocation to lower priority pools. As jobs complete, resources are redistributed among the remaining jobs for execution. Other than

leveraging the capabilities of the Hadoop fair-scheduler, no modifications were made to the Hadoop software.

ACCORD's client-cluster protocol includes a message set supporting prioritization, load balancing, and status monitoring of jobs by clients. A client that has a computing job to submit to the cloud initiates the process by requesting a congestion price for the job from all clusters that the client can reach. (Client access can be limited to some subset of otherwise reachable clusters via configuration.) The client evaluates the congestion price from all clusters that return a congestion price before a timeout.

ACCORD has multiple functions to select cluster(s), taking account of the returned congestion price. The simplest function just submits the job to the cluster with the lowest congestion price. The function *compute_expected_utility* uses congestion price, utility of the job, and probability of cluster failure to compute the net expected utility (NEU) of submitting a job to a particular cluster. The cluster with highest NEU is selected. Use of the NEU concept to control replication of jobs or tasks across clusters is explored in Section V.

The client-cluster protocol is comprised of *price_query*, *price_response*, *job_submit*, *job_complete* and *job_status* messages. A client sends *price_query* messages to available clusters and waits for *price_response* messages. Clusters compute the congestion price based on instantaneous load, resources required to finish the job and reply with a congestion price for the job. Clients evaluate all congestion prices and send a *job_submit* message to cluster(s). The winning cluster(s) periodically sends *job_status* messages to the client with estimated completion time. The cluster sends a *job_complete* message when the job is finished.

Currently, the clusters always reply with the instantaneous congestion price regardless of the number of *price_query* messages it had received. This aspect of the client-cluster protocol can cause a relatively unloaded cluster to become temporarily overloaded, because its advertised congestion price only reflects the load of existing jobs, while ignoring potentially numerous, imminent job submissions.

One possible approach to address this issue is to make congestion prices available to all clients reachable from the cluster (e.g., via a publish-subscribe mechanism). The cluster will publish congestion prices periodically and asynchronously as they change. The clients will choose the cluster and submit the job based on this asynchronous price information, where the job submission indicates the price assumed by the client. The cluster may choose to accept the job if its instantaneous congestion price matches the price indicated in the job submission, or reject the job if the prices differ significantly. The client will then repeat the job submission process until one of the clusters accepts the job.

IV. Testbed and Experimental Results

Our testbed consists of clients that can be scripted to submit jobs at random times and at designated priority levels. The network consists of a series of routers providing configurable connectivity to the clusters. Each cluster runs an instance of Hadoop and the ACCORD cluster controller software. Impairments to the network such as limited bandwidth, packet loss, and loss of connectivity can be applied to the network in an automated fashion. We used three clusters of different computational capacities in our experiments. Each cluster functioned independently, including NBS and congestion price computation.

We show our experimental results for four scenarios. Scenario 1 focuses on Nash Bargaining within a single cluster to demonstrate the proportional fair sharing of computing resources among prioritized jobs within the cluster. Scenario 2 shows how Nash Bargaining successfully

balances prioritized loads among the three clusters within our testbed. Scenario 3 illustrates a failure situation where a single micro-cluster fails. Via Nash Bargaining and congestion pricing, clients redistribute their loads among the remaining clusters. Scenario 4 illustrates a case where a cluster fails but is later restored and becomes accessible to clients.

IV.A. Scenario 1

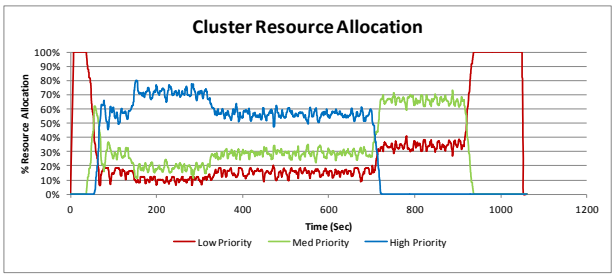

Figure 3 - Single Cluster Resource Allocation via NBS.

Figure 3 shows the fair proportional sharing of computing resources within a single cluster. Each of 3 clients submitted a job at a staggered interval. The jobs were long in duration and consisted of many Hadoop "map" tasks. Each client submitted its job at a different priority level. Priorities were structured such that the medium-priority job was weighted twice that of the low-priority job, and the high-priority job was weighted at twice that of the medium-priority job. Jobs were submitted from lowest priority to highest priority.

The graph shows the percentage of resources allocated to each priority level as a function of time. Since the cluster was initially idle, all cluster resources were initially allocated to the lowest priority job. Fair proportional scheduling allows resources to be consumed by lower-priority jobs when no other competing jobs exist. At approximately 50 seconds, clients submitted the 2 higher-priority jobs. As can be seen the cluster computing resources were redistributed among the jobs proportionally to the priorities of each job – roughly 58%, 28% and 14% of resources to each of the priority levels.

A second, shorter high-priority job was submitted by a fourth client approximately 150 seconds into the run. The system rebalanced the computing resource proportionally, based on the new job mix – roughly 70% to the high-priority jobs, 20% to the medium-priority job, and 10% to the low-priority job. Rebalancing was accomplished by reassigning the weights via the ACCORD cluster controller assigned to each pool to reflect the current job priority mix. Although the number of tasks belonging to high-priority jobs was large, the lowest-priority job was never starved for resources. Once the shorter high-priority job completed, the system rebalanced to the earlier state. When the highest-priority job completed, the computing resources were rebalanced, allocating 2/3 of the computing resources to the medium-priority job and 1/3 to the lowest-priority job, reflecting the priority weightings originally assigned to the pools. Eventually all resources were allocate to the low-priority job. The system continuously rebalanced its resources based on the quantity and priority levels of the jobs being executed.

Figure 4 shows the resource allocations from a sequence of short jobs each comprising a relatively few tasks of short duration. Each of the 3 clients submitted their sequence of jobs at an average of 10 second intervals at a specific priority level. The balancing of computing resources is similar to that shown in Figure . Since the system favored the higher priority client, the

computing resources allocated to this client reached a steady completion rate and was able to keep up with the arrival rate of the high-priority client. However, since the computing resource applied to the lower-priority jobs was reduced, the number of lower-priority jobs in the system actually increased. Due to proportional fair sharing, the ratio of lower-priority jobs to high-priority jobs increased. The net result was that more computing resources were allocated to the lower-priority jobs. At 250 seconds, the high-priority client doubled its job submission rate. Due to Nash Bargaining, the computing resources were rebalanced proportionally providing similar resources to that shown in Figure . Once jobs completed, resource were rebalanced accordingly.

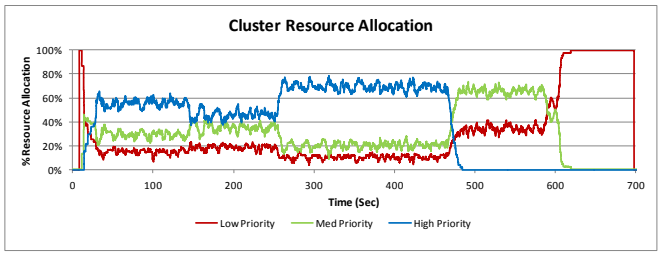

Figure 4 - Single Cluster Resource Allocation - Short Jobs

Figure 5 shows that NBS allocation of resources within the cluster consistently enabled higher-priority jobs to finish faster than lower-priority jobs concurrently executed in the cluster.

Figure 5 – Scenario 1 Job Duration

IV.B. Scenario 2

Scenario 2 focused on multiple clusters and the use of congestion prices for cluster selection. In this scenario, clients solicited congestion prices from each cluster prior to job submission. The cluster offering the lowest congestion price was used by the client. No job replication across clusters was included in this scenario. Jobs were of short duration. Clients offered jobs to the clusters at a 10 second inter-arrival rate.

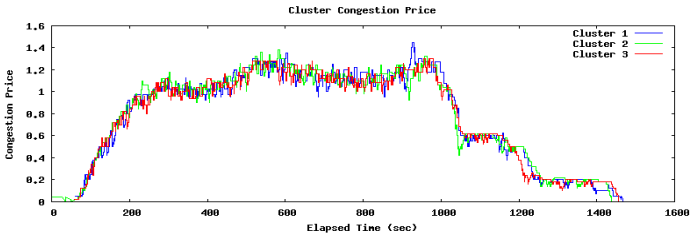

Fig. 6. Congestion price across three clusters in Scenario 2.

Figure 6 shows the results of the use of congestion price to balance loads via ACCORD's distributed NBS implementation. Since congestion price directly reflects the computing load within a cluster, we see that congestion pricing among clusters is consistent. Service rate as reflected by job duration shown in Fig. 7 is also consistent among clusters. Jobs at each priority level achieve similar performance among clusters with that of the highest-priority jobs completing faster than those of lower-priority jobs.

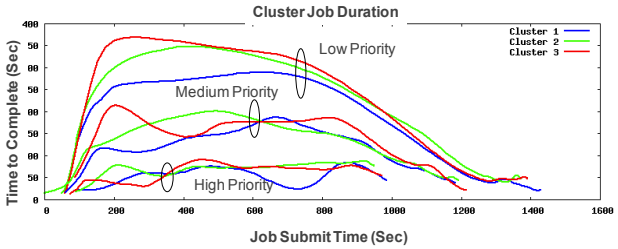

Figure 7 Cluster Job Duration for each cluster and priority level in Scenario 2.

IV.C. Scenario 3

Scenario 3 shows ACCORD's adaptation and resilience in the event of a failed or unreachable cluster. The experiment was similar to that of Scenario 2. However, in Scenario 3, Cluster 1 failed after 600 seconds. Clients submitting jobs detected the failure via time out mechanisms built into their ACCORD price query modules. Figure 8 shows the congestion prices as a result of the failed cluster. Immediately, clients began to submit their jobs to the remaining two clusters. Outstanding jobs not acknowledged by the failed Cluster 1 were also resubmitted to the remaining clusters for execution. The failed cluster caused the congestion price to rise in the respective remaining clusters. However, the computing load rebalanced absorbing the added requests for computing resources to fill the computing resource gap caused by the failure of Cluster 1. Fair sharing of computing resources among the two remaining clusters continued after the failure.

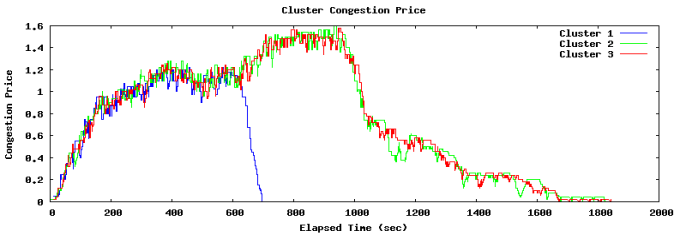

Figure 8 Scenario 3 Congestion Price before and after failure of Cluster 1

Figure 9 shows measurements of Scenario 3 job duration for each of the priority levels. Queuing delays build up due to the reduced overall computing resources as a result of the cluster failure. Relative times for job durations remain consistent and those of high-priority jobs complete before those of lower-priority jobs.

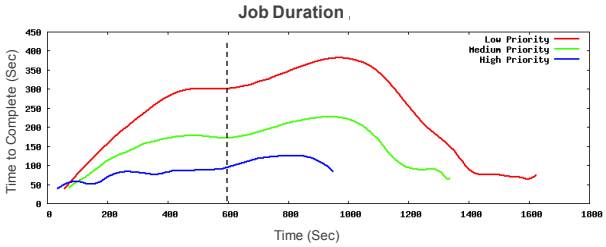

Figure 9 – Scenario 3 Job Duration

IV.D. Scenario 4

Figure 10 shows a cluster recovery scenario. A cluster itself might not fail, but can become unreachable due to sustained network impairment. Clients monitor the status of submitted jobs via the client-cluster protocol described in Section III, and conclude that a cluster and associated jobs have failed after a configurable interval in which no status messages have been received. Clients continue to submit price queries to all known clusters, including those that may be temporarily unreachable. Figure 10 illustrates a case where, at 300 seconds, Cluster 1 became unreachable via the client-cluster protocol, at which point clients submitted their job requests to the remaining Clusters 2 and 3. At 500 seconds, reachability to Cluster 1 was restored. Since no jobs existed in Cluster 1 at that time, its congestion price was lower than those of the other clusters, and clients begin to direct new job submissions to that cluster. We see that after a relatively short interval, the total load across all clusters again rebalanced.

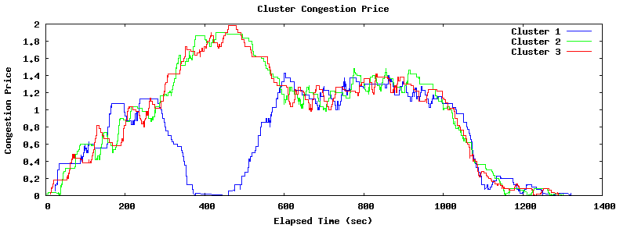

Figure 10 – Re-balancing after cluster failure and restoration.

V. Summary and Further Work

ACCORD provides a means for efficiently and optimally balancing prioritized loads across distributed cloud resources, without the need for centralized, cloud-wide control. The architecture is particularly well-suited to clouds with mobile, wireless clients and resources, but is applicable to any cloud deployment with geographically dispersed cloud capacity, unreliable network access to the cloud, and/or intermittent availability of cloud components. We have shown that a distributed implementation of the NBS provides rapid, adaptive response to attacks or failures within the cloud, with the response maintaining prioritized access to those resources that remain reachable during and after the attack.

Numerous extensions to this initial ACCORD implementation are underway. The first extension focuses on optimal replication of jobs or tasks across clusters. Although the NBS approach in Section II provides an expected utility for a given job instance, and reflects the cost of resources for that instance, it does not take into account the expected cost of failure, or the cost of resources that might be invested to reduce it by running multiple instances of the job in parallel. The idea of adding job or task redundancy within datacenters to overcome isolated cluster component failures has been well explored in current art [5]. However, ACCORD focuses on a different failure mode – complete loss of one or more clusters – that, in the envisioned tactical cloud, may be more prevalent than failure of internal cluster components, due to a variety of failure and attack modes described earlier. Job or task replication within a cluster cannot address this failure mode. Instead, given the probabilities that job instances will fail, ACCORD clients will maximize NEU by estimating the marginal gain in NEU that would result in running redundant job instances at different clusters, whenever multiple clusters are reachable. This NEU maximization trades the reduction in expected cost of failure against the added cost of resources to run the job replica(s).

A second extension concerns maximization of deadline-dependent utility (DDU). Due to the proportional fairness that is inherent in the NBS formulation of Section II, cloud resource allocations that are unaware of job deadlines may not guarantee cloud resources that are sufficient to meet those deadlines (see also [6,7]) As our next step, we plan to investigate the maximization of DDU under the NBS framework in Section II, which aims to assign cloud resources based on not only how much resources are requested by job instances, but also when the resources need to be delivered to ensure mission effectiveness. The DDU maximization shall apportion cloud resources among various missions while offering completion guarantee to time-critical and high-priority jobs.

Acknowledgment

This work was supported by Contract FA8750-11-C-0254 with the US Defense Advanced Research Projects Agency and the Air Force Research Laboratory.

References

[1] S. Kandula *et al*., "The nature of datacenter traffic: measurement and analysis," *Proc. IMC'09*, Chicago, Nov. 4-6.
[2] J. Nash, "The Bargaining Problem", *Econometrica*, Vol. 18, No. 2, April 1950, pp. 155-162.
[3] J. Nash, "Two-person Cooperative Games", Econometrica, Vol. 21, No. 1, April 1953, pp.

128-140.
[4] Tian Lan, David Kao, Mung Chiang, and Ashutosh Sabharwal. "An Axiomatic Theory of Fairness for Resource Allocation", *In proceedings of IEEE INFOCOM*, March 2010.
[5] VMware, "Protecting Mission-Critical Workloads with VMware Fault Tolerance," Technical Report, available online at www.vmware.com/files/pdf/resources/ft_virtualization_wp.pdf, February 2009.
[6] P. Patel, A. Ranabahu, and A. Sheth, "Service Level Agreement in Cloud Computing, in Proceedings of the Workshop on Best Practices in Cloud Computing, 2009.
[7] Wei Wang, Peng Zhang, Tian Lan, and Vaneet Aggarwal. "Datacenter Net Profit Optimization with Individual Job Deadlines", In proceedings of CISS 2012, March 2012.