# CometPortal: A Portal for Online Risk Analytics Using CometCloud

Hyunjoo Kim, Moustafa Abdelbaky, Manish Parashar
*NSF Center for Autonomic Computing*
*Department of Electrical & Computer Engineering*
*Rutgers, The State University of New Jersey*
*94 Brett Road, Piscataway, NJ, USA*
*Email: {hyunjoo, deesha, parashar}@cac.rutgers.edu*

***Abstract*—This paper describes CometPortal, a pervasive portal for accessing cloud computing services provided by CometCloud. CometCloud supports policy-driven, robust autonomic cloud bridging and autonomic cloudbursts. CometPortal provides an interface for monitoring and controlling application deployment using CometCloud, specifying and modifying policies controlling scale-out based on load dynamics, performance requirements and/or economic constraints. Online risk-analytics is used as a driving application to demonstrate the design and operation of CometPortal. Specifically, CometPortal is used to support a cloud-based online Value-at-Risk (VaR) application that is deployed on a virtual cloud combining a private cloud at Rutgers University and the Amazon EC2 public cloud.**

***Keywords*-Cloud computing, Autonomic computing, Autonomic cloudbursts, Autonomic cloud bridging, Portals**

## I. Introduction

Cloud services represent an emerging computing paradigm that is based on-demand access to computing utilities. It essentially provides an abstraction of unlimited computing resources, and supports a usage-based payment model where users essentially "rent" virtual resources and pay for what they use. Integrating these public cloud platforms with existing computational infrastructures provides opportunities for on-demand scale-up, scale-down and scale-out, i.e., cloudbursts. While such a paradigm can potentially have a significant impact on a wide range of application domains, various aspects of the existing applications and of the current cloud infrastructure make the transition to clouds challenging.

CometCloud [1] is designed to address these challenges. It is an autonomic cloud engine that enables applications on virtual cloud infrastructures and support policy-driven robust autonomic cloud bridging (on-the-fly integration of local computational environments (data centers) and public cloud services) and autonomic cloudbursts. Specifically, policies are used to manage dynamically changing application objectives (performance, budgets) and workloads, and define its scale out/in behaviors as well as manage application execution in the virtual cloud environments. This paper describes the design and implementation CometPortal, an essential complement to CometCloud, which provides users with a easy-to-use and pervasive portal for accessing and controlling CometCloud services, defining and modifying policies, accessing cloud resources, and deploying and managing application execution.

The design and deployment of CometPortal is driven by a cloud-based online risk analytics application using Value-at-Risk (VaR). VaR application is a market standard risk measure used by senior management and regulators to quantify the risk level of a firm's holdings. A VaR calculation will typically start after the end of the trading day, and must complete and update risk numbers before the start of the next trading day. As the number and complexity of positions change, the computational requirements for the calculation can change significantly and become highly irregular and dynamic, however the completion deadline remains fixed. As a result, the elastic nature of cloud computing resources makes it a very attractive solution for these applications. The dynamically changing workload and time-critical nature of VaR are supported using the autonomic cloud bridging and cloud bursting capabilities of CometCloud and through the CometPortal. Specifically, CometPortal is used to supports a cloud-based Value-at-Risk (VaR) application and provides an interface to monitor workloads, access cloud resources and control execution behaviors and policies.

CometPortal provides two ways for interfacing with CometCloud, an autonomic mode and a manual mode. In this paper, we focus on handling spikes and dips VaR computational loads by dynamically acquiring and releasing resources in a public cloud, and describe a workload-based policy where the virtual cloud scales up or down based on defined upper and lower workload thresholds. CometPortal and the CometCloud infrastructure have been deployed on a combination of a private cloud at Rutgers University and the Amazon EC2 public cloud and have been used to demonstrate the operation and benefits of CometPortal.

The rest of this paper is organized as follows. Section II

The research presented in this paper is supported in part by National Science Foundation via grants numbers IIP 0758566, CCF-0833039, DMS-0835436, CNS 0426354, IIS 0430826, and CNS 0723594, by Department of Energy via the grant number DE-FG02-06ER54857, by a grant from UT Battelle, and by an IBM Faculty Award, and was conducted as part of the NSF Center for Autonomic Computing at Rutgers University. Experiments on the Amazon Elastic Compute Cloud (EC2) were supported by a grant from Amazon Web Services.

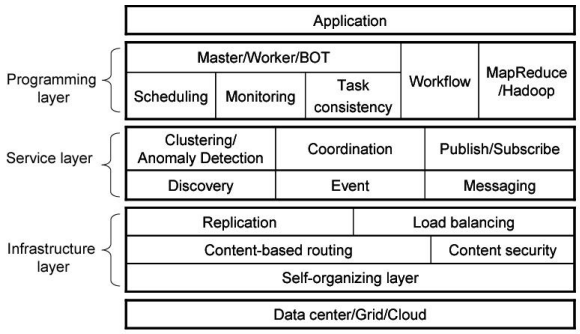



Figure 1. The CometCloud architecture [1].

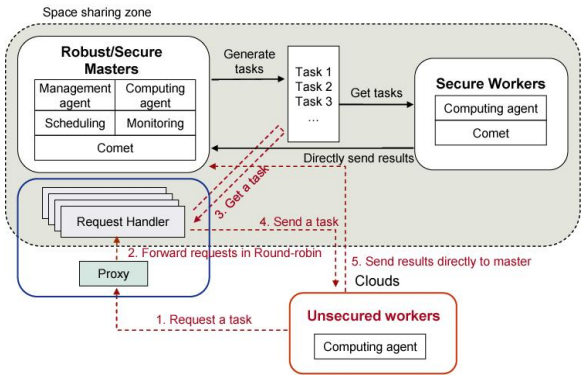



Figure 2. Support for autonomic cloud bursts in CometCloud. [1]

presents an overview of CometCloud. Section III describes the design of CometPortal while its implementation is described in section IV. Section V concludes the paper.

## II. An Overview of CometCloud

### A. Architecture

CometCloud [1] is an autonomic computing engine for cloud and Grid environments. It is based on the Comet [2] decentralized coordination substrate, and supports highly heterogeneous and dynamic cloud/Grid infrastructures, integration of public/private clouds and autonomic cloudbursts. A schematic overview of the architecture is presented in Figure 1.

Conceptually, CometCloud is composed of a programming layer, service layer, and infrastructure layer. The infrastructure layer uses the Chord self-organizing overlay [3], and the Squid [4] information discovery and content-based routing substrate build on top of Chord. The routing engine supports flexible content-based routing and complex querying using partial keywords, wildcards, or ranges. It also guarantees that all peer nodes with data elements that match a query/message will be located. This layer also provides replication and load balancing services, handles dynamic joins and leaves of nodes as well as node failures. Further details about the Comet substrate can be found in [2].

The service layer provides a range of services to supports autonomics at the programming and application level. This layer supports a Linda-like [5] tuple space coordination model, and provides a virtual shared-space abstraction as well as associative access primitives. Asynchronous (publish/subscribe) messaging and event services are also provides by this layer. Finally, online clustering services support autonomic management and enable self-monitoring and control. Events describing the status or behavior of system components are clustered and the clustering is used to detect anomalous behaviors.

The programming layer provides the basic framework for application development and management. It supports a range of paradigms including the master/worker/BOT. Masters and workers can communicate via virtual shared space or using a direct connection. Scheduling and monitoring of tasks are supported by the application framework. The task consistency service handles lost/failed tasks. Even though replication is provided by the infrastructure layer, a task may be lost due to, for example, network congestion. Hence, task consistency is provided for regenerating the lost tasks. If the master receives duplicate results for a task, it selects one (the first) and ignores other (subsequent) results. A leasing mechanism is used for garbage collection. Other supported paradigms include workflows as well as Mapreduce [6] (and Hadoop [7]).

### B. Autonomic Cloud Bridging and Cloudbursts in CometCloud

CometCloud seamlessly (and securely) integrates private enterprise clouds and data centers with public utility clouds on-demand, to provide an abstraction of resizable computing capacity that is driven by user-defined high-level policies. It enables the dynamic deployment of application components, which typically run on internal organizational compute resources, onto a public cloud to address dynamic workloads, spikes in demands, economic/budgetary issues, and other extreme requirements.

The overall support for autonomic cloudbursts in CometCloud is presented in Figure 2. CometCloud considers three types of clouds based on perceived security/trust and assigns capabilities accordingly. The first is a highly trusted, robust and secure cloud, usually composed of trusted/secure nodes within an enterprise, which is typically used to host masters and other key (management, scheduling, monitoring) roles. These nodes are also used to store state. In most applications, the privacy and integrity of critical data must be maintained, and as a result, tasks involving critical data should be limited to cloud nodes that have required credentials. The second type of cloud is one composed of nodes with such credentials, i.e., the cloud of secure workers. A privileged Comet space may span these two clouds and may contain critical data, tasks and other aspects of the application-logic/workflow. The final type of a cloud consists of casual

workers. These workers are not part of the space but can access the space through the proxy to obtain (possibly encrypted) work units as long as they present required credentials and these credentials also define the nature of the access and type of data that can be accessed. Note that while nodes can be added or deleted from any of these clouds, autonomic cloudbursts primarily target worker nodes, and specifically worker nodes that do not host the Comet space as they are less expensive to add and delete.

## III. Comet Portal

An overview of the operation of the CometPortal and the CometCloud on virtually integrated cloud is presented in Figure 3. CometPortal is a java stand-alone application that is used to communicate with the Comet server, and forwards requests from users to the server. These requests include launching or terminating masters and workers, monitoring their state and resources, connecting to a specific node, as well as displaying or graphing results. CometPortal also supports a web-start feature to allow users to interact with CometCloud using a web browser. In order to allow communication between the CometPortal and CometCloud, the Comet server has to be started first to listen for incoming messages from the CometPortal. When the CometPortal starts, it connects to the Comet server first before receiving any inputs from the user. CometPortal provides four key capabilities to manage CometCloud applications, i.e., *Comet initialization, Run application, Apply changes*, and *Stop application*. *Comet initialization* starts a bootstrap node, launches masters and workers, and then waits for them to join the overlay. *Run application* starts generating tasks on masters. Once the generated tasks are inserted into the Comet space, workers can retrieve these tasks through the proxy and consume them. *Apply changes* is for applying changes made by the users such as modifications to the workload policy or changes to the cloud configuration. Note that CometPortal provides two ways for interfacing with CometCloud, an autonomic mode where adaptations and reconfigurations are policy driven and automatic, and a manual mode where the user can explicitly define these behaviors. In the autonomic mode, the Comet server allocates or releases unsecured workers according to workload changes and based on the workload policy. *Stop application* causes a stop message to be sent to the Comet server and the server then forwards this message to all the nodes running the application causing them to terminate.

Once the CometPortal starts up it connects to the Comet server, and obtains connection information for all connected nodes. When the application starts, separate threads running in the background monitor the status of masters as well as secured and unsecured workers. Other threads are used to check for results, and graph them when they are ready. Multi-threading allows a user to interact with CometPortal without waiting for a certain request to be completed. The

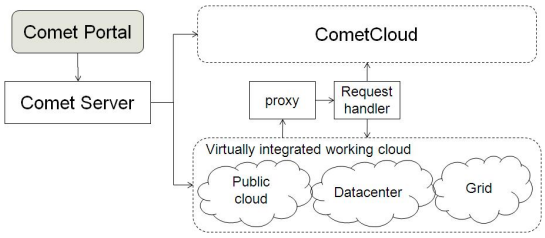


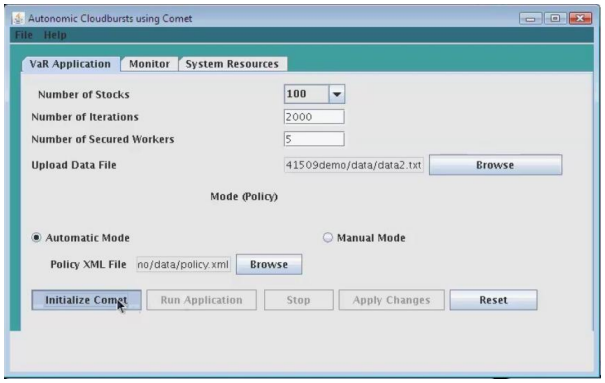

Figure 3. Overview of the medical image registration application scenario using CometCloud.

CometPortal also provides a functionality to view system resources, including CPU and memory for each running machine. A request for system resources goes to the Comet server and the server retrieves this information, which is then sent back to CometPortal. Note that CometPortal does not connect to each machine directly for security reason, but gets the information through the Comet server instead. However, if additional information is required, CometPortal provides the ability to connect to a selected machine using *ssh* protocol (i.e., MindTerm's [8] SSH java implementation from AppGate [9]).

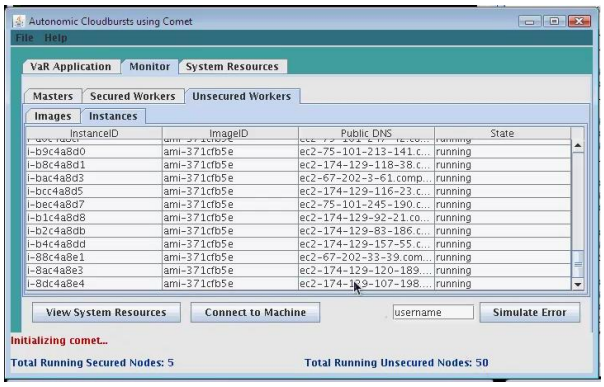
## IV. Implementation

We implemented CometPortal using Java 1.6 and deployed it on the combination of a private cloud at Rutgers University and a public cloud at Amazon EC2. We ran VaR using up to 25 nodes in Rutgers cluster and 100 nodes in EC2. For the policy-based autonomic cloudbursts, we set the upper workload threshold to 2500 scenarios (the number of simulations) and the lower workload threshold to 1500 scenarios. We set the node allocation unit to 50, i.e., the number of nodes allocated when the number of VaR scenarios increases beyond the upper threshold. Further, we set the node release unit to 20, i.e., the number of nodes released when the number of VaR scenarios decreases below the lower threshold. Initially we ran VaR with 50 unsecured workers in the autonomic mode. Figure 4 (a) shows the CometPortal initial input screen. The input screen contains two sections, an application section and a section for autonomic cloudbursts. The application section allows specification of the number of stocks, iterations, secured workers and data file for the VaR application. Here, the number of iterations is the number of Monte-Carlo scenarios that defines the workload. In the autonomic cloudbursts section, a user can select either the autonomic mode or the manual mode. To select the autonomic mode, the policy to be used must be specified. A predefined policy can be selected or a custom policy can be defined.

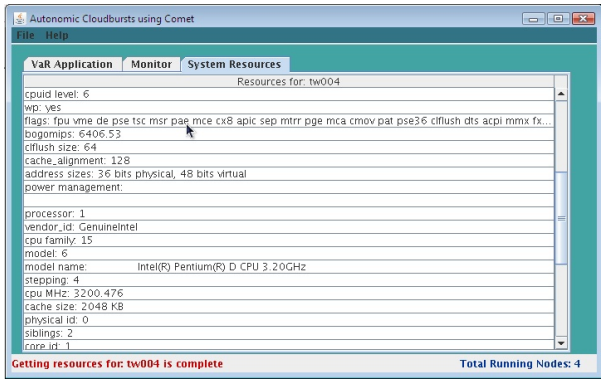
The policy is specified as an XML file. In the case of workload-based policy, it specifies the upper and lower thresholds for workload-based adaptation as well as the allocation and release units. In the manual mode, the number of unsecured cloud workers can be manually set at any time.




(a) Main screen for getting VaR inputs and selecting autonomic cloudbursts




(b) Monitoring screen checking the states of running nodes, and for viewing system resources and connecting a specific node




(c) System resources view including CPU and memory for a specific node


(d) The result of VaR application.

Figure 4. Screenshots of CometPortal.

Fig. 4 presents a CometPortal screenshot showing the list of nodes including masters, secured and unsecured workers. Fig. 4 also presents a screenshot of the interface for viewing system resources and for connecting to a specific machine.

## V. Conclusion

In this paper we presented the design and implementation of CometPortal, a pervasive portal interface that enables access to cloud services provides by CometCloud. CometCloud is a cloud engine that enables applications on virtual cloud infrastructures through robust autonomic cloud bridging and autonomic cloudbursts. CometPortal allows the user to control the dynamic scale-out/in behaviors of CometCloud, and enables monitoring and management of the application and the resources. CometPortal provides support for configuring autonomic policies and operating modes, launching and monitoring VaR applications as well as accessing and monitoring cloud resources. CometPortal and the CometCloud infrastructure have been deployed on a combination of a private cloud at Rutgers University and the Amazon EC2 public cloud and have been used to demonstrate the operation and benefits of CometPortal.

## References

[1] H. Kim, S. Chaudhari, M. Parashar, and C. Marty, "Online risk analytics on the cloud," in *Cluster Computing and the Grid, 2009. CCGRID '09. 9th IEEE/ACM International Symposium on*, May 2009, pp. 484–489.

[2] Z. Li and M. Parashar, "A computational infrastructure for grid-based asynchronous parallel applications," in *HPDC '07: Proceedings of the 16th international symposium on High performance distributed computing*. New York, NY, USA: ACM, 2007, pp. 229–230.

[3] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup protocol for internet applications," in *ACM SIGCOMM*, 2001, pp. 149–160.

[4] C. Schmidt and M. Parashar, "Squid: Enabling search in dht-based systems," *J. Parallel Distrib. Comput.*, vol. 68, no. 7, pp. 962–975, 2008.

[5] N. Carriero and D. Gelernter, "Linda in context," *Commun. ACM*, vol. 32, no. 4, pp. 444–458, 1989.

[6] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[7] Hadoop. http://hadoop.apache.org/core/.

[8] Mindterm. http://www.appgate.com/index/products/mindterm/.

[9] appgate network security. http://www.appgate.com/.