A Cost-efficient Approach for Dynamic and Geographically-diverse Replication of Components in Cloud Computing

[1]N.Pughazendi, [2]C.Bhanu prakash
[1]Faculty, M.C.A, Panimalar Engineering College, Chennai
[2] P.G Scholar, M.C.A, Panimalar Engineering College, Chennai

***Abstract-* A cost-efficient approach for dynamic and geographically-diverse replication of components in a cloud computing infrastructure that effectively adapts to load variations and offers service availability guarantees .In our approach, inter-dependencies(traffic and workflow) among components, their processing overhead and server capabilities are implicitly taken into account by means of server rent prices .we intend to explore our economic paradigm for the self-tuning in the cloud of service components with heavy data dependencies**.

Keywords: components ;net benefit; replication; agent; webservice

I. Introduction

A successful online application should be able to handle traffic spikes and flash crowds efficiently. Moreover, the service provided by the application needs to be resilient to all kinds of failures (e.g. software stales, hardware, rack or even datacenter failures, etc.). A naive solution against load variations would be static over-provisioning of resources, which would result into resource underutilization for most of the time. Resource redundancy should be employed to increase service reliability and availability, yet in a cost effective way. Most importantly, as the size of the cloud increases its administrative overhead becomes unmanageable.
The cloud resources for an application should be self-managed and adaptive to load variations or failures.
Cloud computing is a pay-per-use model for enabling available, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model promotes availability.
However, with static resource allocation, a cluster system would be likely to leave 50% of the hardware resources (i.e. CPU, memory, disk) idle, thus baring unnecessary operational expenses without any profit (i.e. negative value flows). Moreover, as clouds scale up, hardware failures of any type are unavoidable

II. Literature Survey

Literature survey is the most important step in software development process. Before developing the tool it is necessary to determine the time factor, economy n company strength. Once these things r satisfied, ten next steps is to determine which operating system and language can be used for developing the tool. Once the programmers start building the tool the programmers need lot of external support. This support can be obtained from senior programmers, from book or from websites. Before building the system the above consideration r taken into account for developing the proposed system.
We have to analysis the Cloud Computing Outline Survey:
Cloud computing providing unlimited infrastructure to store and execute customer data and program. As customers you do not need to own the infrastructure, they are merely accessing or renting; they can forego capital expenditure and consume resources as a service, paying instead for what they use.

Key characteristics:
Agility improves with users' ability to rapidly and inexpensive lyre-provision technological infrastructure resources.
Application Programming interface(API) accessibility to software that enables machines to interact with cloud software in the same way the user interface facilitates interaction between humans and computers. Cloud computing systems typically use rest-based APIs.
Cost is claimed to be greatly reduced and in a public cloud delivery model capital expenditure is converted to operational expenditure. This ostensibly lowers barriers to entry, as infrastructure is typically provided by a third-party and does not need to be purchased for one-time or infrequent intensive computing tasks.
Device and location independence enable users to access systems using a web browser regardless of their location or what device they are using (e.g., PC, mobile phone). As infrastructure is off-site (typically provided by a third-party) and accessed via the Internet, users can connect from anywhere.
Multi-tenancy enables sharing of resources and costs across a large pool of users thus allowing for:
Centralization of infrastructure in locations with lower costs (such as real estate, electricity, etc.)
Peak-load capacity increases (users need not engineer for highest possible load-levels)
Reliability is improved if multiple redundant sites are used, which makes well designed cloud computing suitable for business continuity and disaster recovery.
Scalability via dynamic ("on-demand") provisioning of resources on a fine-grained, self-service basis near real-time, without users having to engineer for peak loads. Performance is monitored and consistent and loosely coupled architectures are constructed using web services as the system interface.
Security could improve due to centralization of data, increased security-focused resources, etc., but concerns can persist about

loss of control over certain sensitive data, and the lack of security for stored kernels. Security is often as good as or better than under traditional systems, in part because providers are able to devote resources to solving security issues that many customers cannot afford. However, the complexity of security is greatly increased when data is distributed over a wider area or greater number of devices and in multi-tenant systems which are being shared by unrelated users. In addition, user access to security audit logs may be difficult or impossible. Private cloud installations are in part motivated by users' desire to retain control over the infrastructure and avoid losing control of information security.

Maintenance of cloud computing applications is easier, because they do not need to be installed on each user's computer. They are easier to support and to improve, as the changes reach the clients instantly.

Cloud Architecture :

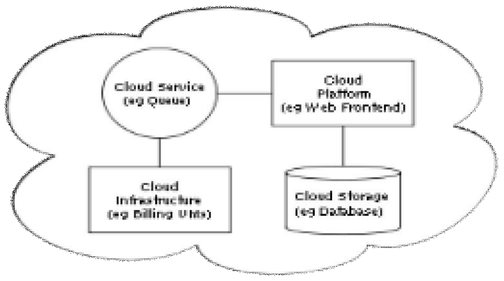
Cloud Service (eg Queue)
Cloud Platform (eg Web Frontend)
Cloud Infrastructure (eg Billing Units)
Cloud Storage (eg Database)

Figure 1.Cloud sample Architecture

Cloud architecture, the systems architecture of the software systems involved in the delivery of cloud computing, typically involves multiple cloud components communicating with each other over application programming interfaces, usually web services and 3-tier architecture .This resembles the Unix philosophy of having multiple programs each doing one thing well and working together over universal interfaces. Complexity is controlled and the resulting systems are more manageable than their monolithic counterparts.

The two most significant components of cloud computing architecture are known as the front end and the back end. The front end is the part seen by the client, i.e. the computer user. This includes the client's network (or computer) and the applications used to access the cloud via a user interface such as a web browser. The back end of the cloud computing architecture is the 'cloud' itself, comprising various computers, servers and data storage devices.

III. SYSTEM ANALYSIS

Existing system:

Successful online application should be able to handle traffic spikes and flash crowds efficiently. Moreover, the service provided by the application needs to be resilient to all kinds of failures (e.g. software stales, hardware, rack or even datacenter failures, etc.). A naive solution against load variations would be static over-provisioning of resources; this would result into resource underutilization for most of the time. Resource redundancy should be employed to increase service reliability and availability, yet in a cost-effective way. Most importantly, as the size of the cloud increases its administrative overhead becomes unmanageable. The cloud resources for an application should be self managed and adaptive to load variations or failures.

Proposed system:

Building an application that both provide robust guarantees against failures (hardware, network, etc.) and handles dynamically a load spike is a non-trivial task. We have developed a simple web application for selling e-tickets (print@home).Composed by 4 independent components:

(i.e.) web front- end, user manager, ticket manager, e-ticket generator.

Each component can be regarded as a stateless, standalone and self-contained web service. Figure 2 depicts the application architecture. A token (or a session ID) is assigned to each customer's browser by the web front-end and is passed to each component along with the requests. This token is used as a key in the key-value database to store the details of the client's shopping cart, such as the number of tickets ordered. Note that even if the application uses the concepts of sessions, the components themselves are stateless (i.e. they do not need to keep an internal state between two requests).

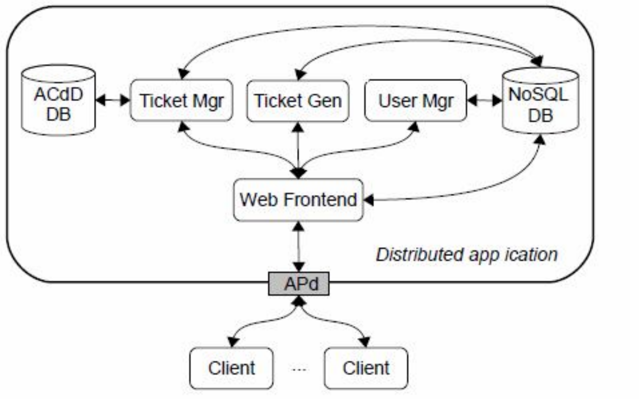

Figure 2 **.**A Distributed Application using different components.

IV. EVALUATION

Experimental Setup:

To evaluate the performance of the proposed application, We employ two different test bed settings: a single application setup consisting of 7 servers and a multi application setup consisting of 15 servers. In the former setup, the cloud resources serve 1 application and in the latter one 3 applications. The hardware specification of each server is Intel Core i7 920 @ 2.67 GHz, 8GBRam, Linux 2.6.32-trunk-amd64. We run two databases MySQL5.1 and Cassandra 0.5.0) as well as one generator of client requests for each application (FunkLoad1.10,http://funkload.nuxeo.org/) on their own dedicated servers Thus, the cloud consists of 4 and 10 servers in the single application and the multi-application setup respectively. We assume that the components of the application may require 1 up to all servers in the cloud.

We simulate the behavior of a typical user of the e-ticket application of Section II by performing the following actions: 1) request the main page that contains the list of entertainment events; 2) request the details of an event A; 3) request the details of an event B; 4) request again the details of the event A; 5) login into the application and view user account; 6 update some personal information; 7) buy a ticket for the event A; 8) download the corresponding ticket in PDF. A client continuously performs this list of actions over a period of 1 minute. An epoch is set to 15 seconds and an agent sends gossip messages every 5 seconds. Moreover, the default routing policy is the random-based policy.

We consider two different placements of the components:

- A static approach where each component is assigned to a server by the system administrator.
- A dynamic approach where all components are started on a single server and dynamically migrate replicate stop according to the load or the hardware failures.

Results:

First, we employ the single-application experimental setup to compare our approach with static placements of the components, where we consider two cases: i) each different component is hosted at a different dedicated server; ii) full replication, where every component is hosted at every server. The response time of the 95% percentile of the requests is depicted in Figure 3. In the static placement (i), where a component runs on its own server, the response time is lower bounded by that of the slowest component (in our case, the service for generating PDF tickets).

Thus, the response time increases exponentially when the server hosting this component is overloaded. In the case of full replication [static placement ii)], the requests are balanced among all servers, keeping the latency relatively low, even when the amount of concurrent users is significant.

In the dynamic placement approach, all components are hosted at a single server at startup: then, when the load increases, a busy component is allowed to replicate, and unpopular components may replicate to a less busy server. Our economic approach achieves better performance than full replication, because the total amount of CPU available in the cloud is used in an adaptive manner by the components: processing intensive (or “heavy”) components migrate to the least loaded servers and heavily used components are assigned more resources than others.

Therefore, the cloud resources are shared according to the processing needs of components and no cloud resources are wasted by over-provisioning.

Also, as the cloud resources are properly utilized by the economic approach, the application throughput (i.e the number of request served per second) that it achieves outperforms static placements, as depicted in Figure4.

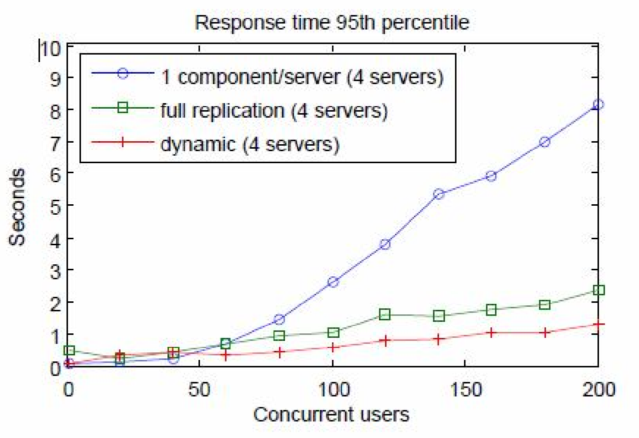

Figure 3. Response time of different placement approaches.

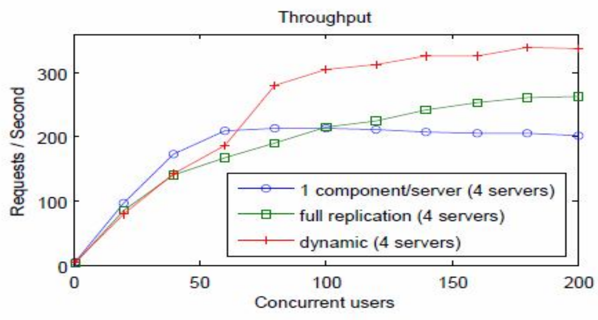

Figure 4. Throughput compared with different placement approaches.

V.IMPLEMENTATION

A web front-end, which is the entry point of the application and serves the HTML pages to the end user.

A user manager for managing the profiles of the customers. The profiles are stored in a highly scalable, eventually consistent, distributed, structured key-value store.

A ticket manager for managing the amount of available tickets of an event. This component uses a relational database management system.

An e-ticket generator that produces e-tickets in PDF format (print@ home).

Server agent

The server agent is a special component that resides at each server and is responsible for managing the resources of the server according to our economic-based approach.

Routing table

A component may be hosted by several servers; therefore we consider 4 different policies that a server s may use for choosing the replica of a component:

a proximity-based policy: thanks to the labels attached to each server, the geographically nearest replica is chosen;

a rent-based policy: the least loaded server is chosen; this decision is based on the rent price of the servers.

a random-based policy: a random replica is chosen.

a net benefit-based policy: the geographically closest and least loaded replica. For every replica of the component residing at server j, we compute a weight.

VI. CONCLUSIONS

Our approach offers high availability guarantees by maintaining a certain number of the various components in geographically diverse locations.We proposed an economic, lightweight approach for dynamic accommodation of load spikes for composite web services deployed in clouds. Application components act as individual optimizers and autonomously replicate, migrate or stop based on their economic fitness. Inter-dependencies (traffic and workflow) among components, their processing overhead and server capabilities are implicitly taken into account by means of server rent prices. As a future work, we intend to explore our economic paradigm for the self-tuning in the cloud of service components with heavy data dependencies.

REFERENCES

[1]“TheApacheCassandraproject,”http://cassandra.apache.org/

[2] L. Lamport, “The part-time parliament,” ACM Transactionson Computer Systems, vol. 16, pp. 133–169, 1998.

[3] N. Bonvin, T. G. Papaioannou, and K. Aberer, “Cost-efficient and differentiated data availability guarantees in data clouds,” in Proc. of the ICDE, Long Beach, CA, USA, 2010.

[4] C. A. Waldspurger, T. Hogg, B. A. Huberman, J. O. Kephart, and W. S. Stornetta, “Spawn: A distributed computational economy,” IEEE Transactions on Software Engineering, vol. 18, pp. 103–117, 1992.

[5] O.Regev and N. Nisan, “The popcorn market. online marketsfor computational resources,” Decision Support Systems, vol. 28, no. 1-2, pp. 177 – 189, 2000.

[6] A. Helsinger and T. Wright, “Cougaar: A robust configurable multi agent platform,” in Proc. of the IEEE Aerospace Conference, 2005.

[7] J. Brunelle, P. Hurst, J. Huth, L. Kang, C. Ng, D. C. Parkes, M. Seltzer, J. Shank, and S. Youssef, “Egg: an extensible and economics-inspired open grid computing platform,” in Proc. of the GECON, Singapore, May 2006.

[8] J. Norris, K. Coleman, A. Fox, and G. Candea, “Oncall: Defeating spikes with a free-market application cluster,” in Proc.of the International Conference on Autonomic Computing, New York, NY, USA, May 2004.

[9] C. Pautasso,T. Heinis, and G. Alonso, “Autonomic resource provisioning for software business processes,” Information and Software Technology, vol. 49, pp. 65–80, 2007.

[10] A. Dan, D. Davis, R. Kearney, A. Keller, R. King, D. Kuebler, H. Ludwig, M. Polan, M. Spreitzer, and A. Youssef, “Web services on demand: Wsla-driven automated management,” IBM Syst. J., vol. 43, no. 1, pp. 136–158, 2004.