Groupdeal Web-service Using Multi-storage and Multi-tier Architecture

Shubhangi Kulkarni, Computer Department, Sinhgad Institute of Technology, Lonavala.
Pooja Jadhav, Computer Department, Sinhgad Institute of Technology, Lonavala.
Dipali Pawar, Computer Department, Sinhgad Institute of Technology, Lonavala.
Dipika Muktule, Computer Department, Sinhgad Institute of Technology, Lonavala.

ABSTRACT
Groupdeal is a web-service that will provide a facility where individual user can buy a product or quantity of products at a wholesale rate. This paper also considers multi-tier architecture which will enhance IOPS performance and storage capacity using replication of data storage that aggregate the disk of many nodes spread over the internet. Transient network or host failure lead to copy data over internet to maintain replication level. Durability can be separately provided from the availability. load balancing and recovery techniques are needed to achieve high performance and availability.

INTRODUCTION
Groupdeal is a web-service which will provide a facility to individual user to buy one or more quantity of products at wholesale rate. Discount will increase as the number of potential users will increase. The following features are comprised in this paper:
1) Multi-tier architecture for QoS
2) Multi-Storage Database system
3) Data replication

I. Multi-tier Architecture

There are generally three layers in client server architecture Presentation layer, application layer, data access layer. Using n-tier architecture developers can create flexible and reusable applications. In the web development field, three-tier is often used to refer to websites, commonly electronic commerce websites, which are built using three tiers:

1. A front-end web server serving static content, and potentially some cached dynamic content. In web based application, Front End is the content rendered by the browser. The content may be static or generated dynamically.
2. A middle dynamic content processing and generation level application server, for example
Ruby on Rails, Java EE, ASP.NET, PHP, ColdFusion, Perl platform.
3. A back-end database or data store, comprising both data sets and the database management system or RDBMS software that manages and provides access to the data.

Using multiple tiers provides a flexible trade-off in terms of IOPS performance and storage capacity, we believe that providing performance isolation and QoS guarantees among various clients, gets significantly more challenging in such environments. we first argue that providing QoS in multi-tiered systems is quite challenging. This allows for higher efficiency of the underlying system while providing desirable performance isolation. We make a case that QoS in multi-tiered storage is an open problem. Storage QoS needs to be aware of the massive performance differences across tiers in order to provide performance isolation while maintaining high efficiency of the underlying devices. In this paper we discuss how the significant differences in speed between in meeting client’s expectations.

II. Multi-Storage Database system
Cloud storage is often implemented with complex, multi-tiered distributed systems built on clusters of commodity servers and disk drives. Management, load balancing and recovery techniques are needed to achieve high performance and availability. Among different failure sources that include software, hardware, network connectivity, and power issues. We study end to end data availability in a cloud computing storage environment which often use loosely coupled distributed storage systems.

The characteristics of such systems are:
1) Storage server programs running on physical machines data center. We refer to the storage server programs as storage nodes.
2) Storage services masters managing data placement, load balancing and recovery, and monitoring nodes.
3) A replication mechanism for user data to provide resilience to individual failures of components. A collection of nodes are called a cell or storage cell. A cell may comprise many thousands of nodes housed in a single building.

A storage node becomes unavailable when it fails to respond positively to periodic health checking pings. Nodes can become unavailable for different reasons. A storage node can be overloaded; a operating system may crash or restart; a machine may experience a hardware problem; repair processes may remove disks or machines; or the whole cluster could be brought down for maintenance temporarily.

III. Data Replication
Distributed storage systems increase chances of failures by using replication. In both cases, data is divided into a set of stripes, which includes a set of fixed size data and code blocks which are called chunks. Data in a stripe can be recovered from some subsets of the chunks. For replication, R = n refers to n similar chunks in a stripe, so the data may be recovered from any one of the chunks.

A. Data replication and recovery
Replication prevent node failures. When a node failure causes the non-availability of a chunk within a stripe, we start a recovery operation for that chunk from the other available in the stripe. Distributed files will employ queues for recovery operations following node failure. These queues help in reconstruction of stripes which have lost the most chunks. The rate at which missing chunks may be recovered depends upon the bandwidth of disks, nodes, and racks. There is an exclusive design trade off in the use of bandwidth for recovery operations versus serving client read/write requests.

B. Durability And Availability
Durability means a object that an application has put into the system is not lost due to disk failure and availability means that get will be able to return the object promptly. If the only copy of an object is on the disk of node that is currently powered off, but will someday re-join the system with disk, then that object is durable but not currently available. Durability algorithm must be to create new copies of data objects faster than permanent disk failures destroy the objects; careful choice of policies for what nodes should hold data and take less repair time. Increasing the number of replicas does not help a system tolerate a higher disk failure probability, it helps to tolerate bursts of failures.

C. Carbonite Algorithm
The Carbonite maintenance algorithm on reintegration to avoid responding to transient failures. RL is Durability; It should place objects to maximize q and repair the least replicated object. It works to efficiently maintain RL copies, thus providing durability. Because it is not possible to distinguish between transient and disk failures remotely, It simply responds to any detected failure causes the non-availability of a chunk. A replication mechanism for user data to provide resilience to individual failures of components.
stored on which node that have failed so that they can be reused if they return. It will greatly reduce the cost of responding to transient failures.

// Iterate through the object database
// and schedule an object for repair if needed

MAINTAIN_REPLICAS ()
keys = <DB.object_keys sorted by number of available replicas>
for each k in keys:
    n = replicas[k].len ()
    if (n < rL)
        newreplica = enqueue_repair (k)
        replicas[k].append (newreplica)

Using synchronisation mechanism each node maintain a list for which it is responsible and monitor replication level. In this code as per implementation it should store on disk but it store in replicas hash table. This code is called periodically to enqueue repairs on those objects that have too few replicas available; the application can issue these requests at its convenience.

For example, if the system has created two replicas beyond rL and both fail, no work needs to be done unless a third replica fails before one of the two currently unavailable replicas returns. Once enough extra replicas have been created, it is unlikely that fewer than rL of them will be available at any given time. Over time, it is increasingly unlikely that the system will need to make any more replicas.

IV. CONCLUSION
To provide high performance and high availability for web-based service. We used multi-tier architecture to improve QoS and multi-storage data, data replication to recover from transient network or host failures. Inexpensive hardware and the increasing capacity of wide-area network links have spurred the development of applications that store a large amount of data on wide-area nodes. However, the feasibility of applications based on distributed storage systems is currently limited by the expense of maintaining data. This paper has described a set of techniques that allow wide-area systems to efficiently store and maintain large amounts of data.

V. REFERENCES


