[JOURNAL OF ADVANCES IN SCIENCE AND TECHNOLOGY
VOL. & ISSUE - I]
February 1, 2011
ISSN-2230-9659

Design the managed and shared common storage system of various applications

Gunjan Goel

Research Scholar, Singhania University

Rajasthan, India

INTRODUCTION

Storage requirements of commercial and institutional organizations are growing rapidly. A popular approach for reducing the cost and complexity of management is to consolidate the separate computing and storage resources of various applications into a common pool. The resources can then be managed together and shared more efficiently. Distributed storage systems, such as Federated Array of Bricks (FAB) and IceCube are designed to serve as large storage pools. They are built from a number of individual storage nodes, or bricks, but present a single, highly-available, storage store to the user.

A typical distributed storage system consists of symmetrical bricks each runs the same set of software modules. Typically, a client accesses the data through a coordinator/gateway, which locates the brick where data resides and transfers the data. In many cases, a brick acts both as a storage center and a coordinator. Different requests, even from the same client, can be coordinated by different bricks. A logical volume in a distributed storage system might be replicated or erasure-coded across many bricks. The goal of performance virtualization here is to design a distributed scheduler that can provide service guarantees regardless of the data layout.

RELATEDWORK

The majority of previous work assume a centralized scheduler exists and clients have the same data layout, e.g., same logical volume. Scheduling methods include fair queuing, real-time scheduling, feedback control. A recent paper guarantees fair service among clients with different data layouts. It uses a centralized scheduler as well. Finally, distributed scheduling of a single resource has been studied in networking domain, where the resource is usually the communication channel.

The problem of fair scheduling that involve both distributed schedulers and different data layouts has not been addressed.

THE APPROACH

If we would like to apply a centralized scheduling algorithm to our distributed architecture, one simple idea would be that coordinators broadcast the information of each request dispatched and the disk back-end enforces an ordinary centralized scheduler. For example, if a coordinator dispatches a request from client f to brick *A*, let it broadcast a *virtual request* of the same content to all other bricks. At the back-end, a virtual request takes no processing time, but does account for the service share allocated to client *f*. Then, taking into account the service cost of these virtual requests, the scheduler can properly allocate resources.

The problem with this idea is of course the communication overhead there might be thousands of bricks in a large-scale data center. Observing that the back-end scheduler needs to know only the total service cost of virtual requests rather than their contents, the coordinator can simply count the total cost and piggyback the information onto the next real request.

The above idea can be well implemented with fair queuing algorithms. A fair queuing scheduler typically maintains a virtual clock (VC) for each client, which advances proportionally with the service cost of the client's requests processed. The scheduler orders requests by their time tags derived from the VC. Start-time Fair Queuing (SFQ) further requires an inactive client to synchronize its VC with active ones, thus guarantees "use it or lose it". When SFQ is adapted to our framework, the scheduler should advance the VC with both the cost of a real request and the total cost of virtual requests.

As SFQ guarantees the service of a client proportional to its weight, our distributed SFQ algorithm guarantees proportional total service across all bricks, whenever possible. Proportional total service may result in no service on some individual brick, this problem can be solved within our framework by reserving a minimum service for each client on each brick.

SUMMARY

ISSN-2230-9659

We developed a novel distributed scheduling framework that can incorporate many existing centralized fair queuing algorithms. Within the framework, we applied Start-time Fair Queuing algorithm for proportional sharing of system throughput. We evaluated the proposed algorithms both analytically and experimentally on a prototype FAB system. The results confirm that the algorithm allocates resources fairly under various settings different data layouts, clients accessing the data through multiple coordinators, and fluctuating service demands.

REFERENCES

- M. Vin, and H. Cheng. Start-time fair queueing: A scheduling algorithm for integrated services packet switching networks.
- IBM. Icecube – a system architecture for storage and internet servers.
  - http://www.almaden.ibm.com/StorageSystems/index.shtml.
- W. Jin, J. S. Chase, and J. Kaur. Interposed proportional sharing for a storage service utility.
- M. Karlsson, C. Karamanolis, and X. Zhu. Triage: Performance isolation and differentiation for storage systems.
- C. R. Lumb, A. Merchant, and G. A. Alvarez. Fac¸ade: virtual storage devices with performance guarantees.
- Y. Saito, S. Frølund, A. Veitch, A. Merchant, and S. Spence. Fab: Building distributed enterprise disk arrays from commodity components.