Existential QoS for Storage

Val Henson
vhenson@eng.sun.com

Jeff Bonwick
bonwick@eng.sun.com

Matt Ahrens
ahrens@eng.sun.com

Sun Microsystems, Inc.
17 Network Circle
Menlo Park, CA 94025

ABSTRACT

In this paper we describe two quality of service (QoS) management models, grammar-based QoS and existential QoS, in the context of storage systems. We compare the two models in terms of generality, implementation complexity, and, most importantly, ease of administration. We describe one implementation of existential QoS, in the Zettabyte File System. We conclude that the existential QoS model provides more flexibility and configurability, eliminates many difficult design problems, and simplifies administration.

1. INTRODUCTION

Administrators want simple and automated management of storage, such as automatic load-balancing across disks and migration of data depending on usage patterns. One way to provide such self-tuning of performance is system management of QoS of storage. This paper will describe two models of managing QoS of storage and compare them in terms of both implementability and complexity of administration. The first QoS model, "grammar-based QoS," lets administrators specify their desired QoS using a grammar that describes a set of qualities and their minimum required levels. Despite its apparent simplicity, this model turns out to be difficult to implement and to use. The second QoS model, "existential QoS," allows storage to be grouped together into a few distinct pools, distinguished by names chosen by the administrator. The QoS of each pool is implicitly defined by the qualities of the pool itself, rather than by a grammar. We find that existential QoS is simpler to implement and use.

2. WHAT IS QOS?

Quality of service is difficult to define. This nebulousness is part of the difficulty of coming up with a manageable QoS framework for storage. For now, we will loosely define QoS of storage to mean any of a number of factors that affect availability, performance, reliability, security, and capacity of storage. We'll discuss QoS in terms of qualities and their values. A **quality** is some attribute of storage, and a **value** is the requested amount or level of a quality. A **QoS setting** is a collection of qualities and their values. A **QoS type** is a level of QoS actually provided by the system.

3. GRAMMAR-BASED QOS

Grammar-based QoS goes like this: Define some reasonable set of qualities, each of which has a number of possible discrete values. The result is a grammar for expressing QoS as a combination of requirements defined by the qualities and their values. The administrator uses this grammar to specify the desired QoS, and the QoS management system then automatically calculates how to provide the level of QoS requested. An example of the grammar-based QoS model, known as attribute-managed storage, is described in [2].

This model seems simple enough on the surface but is actually rather complex. Let's start with defining the set of qualities in a hypothetical QoS grammar. In our experience, many people feel that there are only a few (on the order of 10) qualities of service that are important, but in practice they are unable to come to agreement on which 10 qualities those are or what range of values should be allowed. Even if we assume that we have somehow narrowed down the list to 10 qualities each with 3 values, we now have $3^{10} = 59049$ different QoS settings - a number which should give both administrator and system architect pause. Given our 59049 possible QoS settings, some settings describe unsatisfiable combinations of QoS – not every quality can be maximized without sacrificing any other quality. In the case of an unsatisfiable QoS request, the system has to either give up or attempt to provide the best service possible. Automatically maximizing QoS when the system can't provide the requested QoS is a difficult problem - efficiently finding extrema of functions of many variables (also known as partial constraint satisfaction) is notoriously difficult (specifically, it is NP-complete)[3].

How should the QoS manager handle situations where it can only partially satisfy the QoS requirements? Consider an application that wants both fast disk access and tolerance for one disk failure. The system runs out of capacity on the fast mirrored disk, but it has room on a slow mirrored disk and a fast unmirrored disk. Should the system write to the fast disk, write to the slow mirror, or return an out-of-space error? There is no correct answer for all applications.

The most intractable problem for grammar-based QoS is how to handle shared resources for two applications with conflicting QoS requirements. An example is two applications writing to two different files. One application wants fast, high-bandwidth storage, the other application wants to tolerate one disk failure. The system contains exactly two kinds of storage: one fast new disk, and one slower RAID box. Suppose that both files are in the same directory. Should the directory be stored on the fast disk or the RAID disk or both? Again, there is no correct answer.

In general, the grammar-based QoS model assumes that (a) qualities are orthogonal, (b) each quality varies along a spectrum with "bad" at one end and "good" at the other, (c) all possible necessary qualities are knowable in advance, and (d) when a requested QoS setting is out of range or two applications share metadata, the system can find a compromise that will be reasonably optimal for all applications.

4. EXISTENTIAL QOS

Before describing existential QoS, it will be helpful to briefly outline the terminology and concepts of the Zettabyte File System (ZFS), which is the context for our design and implementation of existential QoS. ZFS is a general purpose POSIX-compliant file system in development at Sun Microsystems with the goals of immense capacity, strong data integrity, and simplified administration. For this discussion, the most relevant feature of ZFS is its pooled storage model. Storage devices are grouped into pools, and file systems dynamically share the space within the pool. By default, each file system within the pool is allowed the full use of all of the resources of the storage devices in the pool: capacity, bandwidth, I/O operations per second, etc. Devices can be added and removed from pools dynamically, and file systems can be migrated between pools.

In the existential QoS model, the QoS of each storage pool is self-describing: the QoS of the storage pool is whatever the pool intrinsically provides as consequence of the physical devices in it To create a pool with a desired QoS, the administrator decides what qualities of service are important, buys the storage devices that will provide those qualities, and creates a storage pool out of the devices. Each storage pool is given a useful, intuitive name such as "archival." We believe that most systems will only require a few different types of QoS, each corresponding to a particular storage pool. In most cases, a user won't be able to distinguish between more than few different types of QoS, and most systems have only a few different kinds of storage attached anway. Jack Gelb makes the same observation in a 1989 paper describing system managed storage[1], which uses the existential QoS model:

> [. . .] The number of storage classes defined in an installation is expected to be small. This is primarily due to the reasonably few distinct levels of service that can be effectively materialized. For example, given eight distinct levels of performance and two distinct levels of availability, a maximum of sixteen (8×2) unique storage classes may be defined. Not all sixteen storage classes may be interesting to a given installation.

Existential QoS allows the administrator to make their own definitions of qualities of service, using their knowledge of what distinctions in QoS are important for the users of a particular system. In the grammar-based QoS model, users would be limited to distinguishing between the pools based on the qualities defined by the QoS manager, and would have to invest significant effort into creating the description of each kind of QoS in the system-mandated QoS grammar. Users may end up painfully reverse-engineering the description of the disks they want to use in order to convince the system to put the correct data on them.

Our QoS granularity (per-file-system rather than per-file) may seem too coarse to be useful. This would be true in a file system where storage can't be shared between file systems. However, in ZFS, file systems are about as cheap and easy to create as directories, since they use only as much storage space as is necessary to store the data they contain.

Existential QoS combined with storage pools addresses the problems we described with grammar-based QoS. The administrator defines what qualities are important, not the system. The system does not need to optimize requested QoS since the administrator chooses which of the few, intuitively named QoS types is most appropriate. Storage pools don't share any resources, so we don't have to solve the problem of shared metadata between objects with different QoS. Existential QoS provides a powerful, general, and above all, simple model for providing QoS to storage system clients.

5. CONCLUSIONS

Grammar-based QoS models introduce an unnecessary, limiting, and cumbersome layer of description between the administrator and the QoS attributes that are actually important for a given system. A small number of qualities and their possible values results in a combinatorial explosion of possible QoS settings, too many for administrators or applications to reasonably choose from. Existential QoS for storage eliminates the difficulty of describing QoS by declaring the QoS of storage to be exactly that QoS which is provided by the underlying hardware. In existential QoS, storage is grouped into a small number of storage pools with intuitive names, such as "financial" or "archival." Each storage pool represents a QoS type. When a file system is created, the administrator selects which storage pool it should draw its storage from. When the QoS needs of that file system change, the file system can be migrated to another storage pool with a more appropriate level of QoS. Shared storage reduces overprovisioning, the QoS types representable are completely unconstrained, and users only have to select from a few different types of QoS.

6. REFERENCES

[1] Jack P. Gelb. System-managed storage. *IBM Systems Journal*, 28(1):77–103, 1989.

[2] Richard Golding, Elizabeth Shriver, Tim Sullivan, and John Wilkes. Attribute-managed storage. In *Workshop on Modeling and Specification of I/O*, 1995.

[3] Zsófia Ruttkay. Constraint satisfaction — a survey. *CWI Quarterly*, 11(2–3):163–214, 1998.