RJ10356 (A0508-013) August 26, 2005
Computer Science

IBM Research Report

Kybos: Self-Management for Distributed Brick-Based Storage

Theodore M. Wong, Richard A. Golding, Joseph S. Glider,
Elizabeth Borowsky*, Ralph A. Becker-Szendy, Claudio Fleiner,
Deepak R. Kenchammana-Hosekote, Omer A. Zaki

IBM Research Division
Almaden Research Center
650 Harry Road
San Jose, CA 95120-6099

*Computer Science Department
Boston College
Chestnut Hill, MA

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Kybos: Self-management for distributed brick-based storage

Theodore M. Wong Richard A. Golding Joseph S. Glider
Elizabeth Borowsky* Ralph A. Becker-Szendy
Claudio Fleiner Deepak R. Kenchammana-Hosekote
Omer A. Zaki
IBM Almaden Research Center, San Jose, CA

Abstract

Current tools for storage system configuration management make offline decisions, recovering from, instead of preventing, performance specification violations. The consequences are severe in a large-scale system that requires complex actions to recover from failures, and can result in a temporary shutdown of the system. We introduce Kybos, a distributed storage system that makes online, autonomous responses to system changes. It runs on clusters of intelligent bricks, which provide local enforcement of global performance and reliability specifications and so isolate both recovery and application IO traffic. A management agent within Kybos translates simple, high-level specifications into brick-level enforcement targets, invoking centralized algorithms only when taking actions that require global state. Our initial implementation shows that this approach works well.

1 Introduction

Large storage systems are complex and expensive to manage. Management tasks often require understanding the internal architecture of the system, and those that make changes involve complex sequence of operations. Some changes require taking storage offline in order to move data, incurring downtime costs. The cost of making changes leads administrators to overprovision systems to accommodate growth without reconfiguration.

Storage management tools have been developed to help solve these problems by automating provisioning and configuration [2, 3, 11]. However, these tools are separate from the systems they manage, making coarse-grained, offline decisions in reaction to recent system behavior. Being separate from the system, they can only observe that a problem has occurred (*e.g.*, a performance violation) and try to reconfigure the storage so it will not recur; they cannot prevent a problem from happening in the first place.

*Computer Science Department, Boston College, Chestnut Hill, MA

Moreover, some require detailed knowledge of the expected application workload to make good decisions, and do not eliminate overprovisioning.

We present the design of Kybos[1], an architecture for a high-performance file system that can scale to petabytes. Kybos is designed for brick-based storage, in which devices that are interconnected via a high-performance network provide storage resources and execute management operations. Bricks provides good scalability through their inherent parallelism, and facilitate simple incremental expansion (to add resources, one just adds bricks). However, a lack of natural central control points makes management mechanisms harder to build.

In Kybos, each brick performs local enforcement of performance and reliability targets to meet global specifications. It reserves resources and shapes IO traffic to isolate applications from each other and from system maintenance actions. A *management agent* within Kybos invokes centralized algorithms only when making decisions that require global state, *e.g.*, when provisioning resources for an application or reacting to failures. In this way the system prevents many kinds of problems, such as performance violations, and reacts quickly when other changes occur, such as brick additions and failures.

We have implemented Kybos as an extension of the IBM SAN.FS (a.k.a. Storage Tank) file system [18]. Some of the results reported are from algorithms that we have investigated using simulations, but have yet to incorporate into the file system.

2 System model

A Kybos cluster is built from bricks. Each brick is a small, self-contained storage server; in our current prototype, each contains commodity CPU, memory, and about 1 TB disk. Bricks have no internal redundancy; however, they connect redundantly to a high-speed, resilient IP network via an internal switch. The internal cluster network forms a rectangular 3D mesh, with multiple external connections to other systems or Kybos clusters [5, 6].

Each brick provides local storage allocation and resource management, so that every brick carries part of the load of maintaining the cluster. It provides local enforcement of security policies using a capability mechanism [9, 12, 21]. Each brick manages local performance, shaping traffic to ensure performance isolation and fair access among IO streams.

Data is stored redundantly across multiple bricks, using a Network RAID protocol to coordinate updates [15]. The protocol uses two-phase writes and client-selected timestamps to implement atomic, serialized operations. The operations are not limited to data read and write, but also include management operations. The timestamp-based approach assists with migrating and rebuilding data concurrently with client IO accesses, and for implementing consistent snapshots.

A management agent monitors bricks for failure, makes global resource allocation decisions, and coordinates migration and rebuild. These decisions require state beyond that held by any one brick.

[1]From the Greek $\kappa\upsilon\beta o\varsigma$ (cube), pronounced “keu-baus”.

3 Virtual entities

A large-scale cluster stores data for several different applications. Each application may use multiple classes of data, with each class potentially having different requirements. For example, a temporary database table may need high performance but only low reliability, while a collection of archived documents may need high reliability but only low aggregate performance.

Kybos provides *resource pools* (RP) to distinguish different classes of data. An administrator specifies requirements for the RP, including capacity, performance, and reliability. Kybos uses these requirements to assign brick resources to back the RP, and determine how to react to events that change resource availability, such as brick addition and failure.

One specifies capacity and performance requirements as reserve/limit tuples. For example, a capacity reserve sets aside storage for an RP, and ensures that no other RP can use that capacity, while the limit places an upper bound on consumption, like a quota. Similarly, a performance reserve is an aggregate guarantee for the streaming throughput of an RP (*e.g.*, the number of concurrent video streams that the underlying bricks for an RP must support), while the limit determines the maximum effect that the streams accessing the RP can have on the system.

A reliability specification includes a redundancy metric, such as the allowed data loss rate, which Kybos translates to a set of distance codes to use for storing data. Kybos does not set aside resources explicitly based on the specification; instead, the specification constrains the assignment of brick resources to RPs, and drives the execution of policies to migrate data.

Files belonging to an application correspond to *virtual objects* (VOs) associated with one of the RPs of the application. As will be explained in §4, the reliability specification for the RP of a VO guides the selection of the distance code for the VO.

The RP and VO virtual entity structure enables Kybos to hide details of how the cluster works, and thus simplify administration. Kybos uses performance and reliability requirements, along with current cluster state, to schedule internal activities to operate the cluster consistent with the requirements. For example, Kybos can respond to addition of a brick by migrating some data onto the brick in order to keep resource usage balanced; the administrator does not need to select data by hand. Virtual entities also insulate the administrator from heterogeneity in bricks.

4 Physical entities

Each of the virtual entities must be backed by physical resources to be useful. In our model, a resource pool (RP) is backed by a set of *allocation pools* (APs), and a virtual object (VO) is backed by a set of *physical objects* (POs). In both cases, we refer to *configuration* as the problem of assigning resources to back virtual entities. Figure 1 below shows the relationship between virtual and physical entities.

We apply online constrained optimization techniques to make configuration decisions. These algorithms make incremental changes to a system, and so support creation and change of RPs as needed. They also enable the reconfiguration of RPs to take ad-

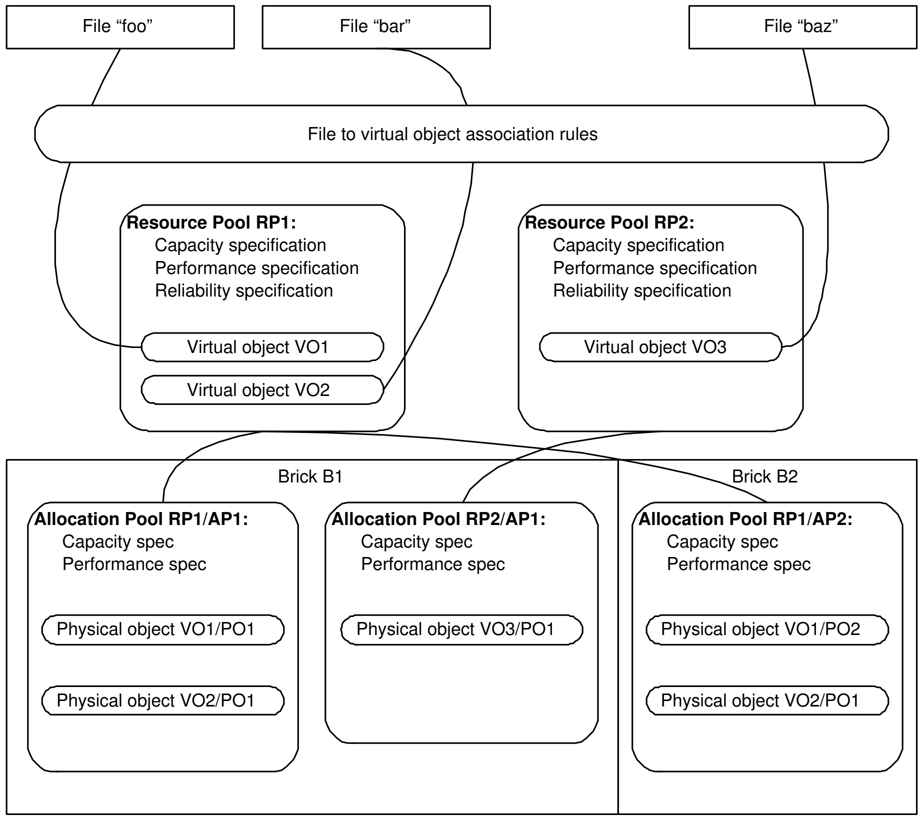

Figure 1: The relationship between virtual and physical entities. RPs are backed on bricks by APs; VOs associated with an RP are backed by POs in APs of the RP.

vantage of the resources on added bricks, or recover from the loss of a brick (§5). Our focus on online techniques differs from other work in the area [2, 3, 11].

An RP is backed by APs, each on a different brick. Each AP has capacity and performance reserve/limit requirements, analogous to RP requirements, that a host brick uses to control capacity usage and shape IO traffic. The APs are subject to the following constraints: the number of APs must be sufficient for the RP reliability requirements; the sum over the AP requirements must equal the RP requirements; and the resources required for each AP must be available on its host brick. Kybos determines the minimum number of APs by computing the minimum distance code that meets the RP reliability requirements, *e.g.*, an RP might specify a reliability that requires at least a distance 2 code, which may yield 2 APs (2-replica RAID-1) or 5 APs (4+1 RAID-5, if desired for better capacity efficiency).

We use a new algorithm, *MinDot*, to configure APs for an RP and place them onto bricks. It determines the appropriate fraction of the RP that should go onto each brick in a way that generally balances load across bricks while respecting configuration constraints, even for clusters of heterogeneous bricks. MinDot is an online algorithm based on a relaxation of the Toyoda zero-one multidimensional bin-packing heuristic [22].

VOs are backed in APs by POs, analogous to how RPs are backed on bricks by APs. The set of POs corresponds to a RAID encoding of the data that meets the reliability requirements of the host RP, *e.g.*, for an RP that requires at least a distance 2 code, the hosted VOs may have 2 POs (2-replica RAID-1), or 5 POs (4+1 RAID-4).

We use different strategies to configure new VOs and to reconfigure existing ones. The POs of a new VO use neither capacity nor performance resources, thus Kybos uses a random-weighted placement strategy that biases placement towards APs with greater free capacity. Later, when Kybos has accumulated resource usage statistics about the POs, it uses a modified version of the Toyoda algorithm [22] (not MinDot) to compute a better configuration that balances AP resource utilization.

5 Reactive self-management

The management agent (§2) monitors the state of the cluster, and takes corrective action whenever it detects anomalous behavior, such as resource usage hot spots, or the addition, removal, or failure of bricks. The actions either *rebalance* or *rebuild* data.

Management actions involve first reconfiguring RPs, and then migrating or rebuilding its hosted VOs to match. Reconfiguring an RP involves re-running MinDot (§4) to compute a new, presumably better, configuration. Reconfiguration may add or remove APs, and change others.

Reconfiguring an RP may result in changes to an AP that are not feasible now, but that will be in future. In this case, the agent sets *goals* on the AP to indicate what the resources should become as local conditions change. For example, an AP might need to shrink usage of some resource; as rebalancing migrates POs to other APs, the actual usage will shrink until it meets the goal. Similarly, an AP might need to grow some resource; as resources become available (*e.g.*, as an AP on the same brick shrinks) they are given to the growing AP until its goals are met.

The management agent periodically rebalances resource usage across bricks. The agent measures balance by the coefficient of variance (CoV—the ratio of the standard deviation to the average of a variable) in resource usage across bricks. The agent alternates in phases between rebalancing VOs and RPs. The agent selects an RP or VO to rebalance, choosing the hottest VO or choosing from any RPs that have goals, and then computes a reconfiguration. The agent ensures that the reconfiguration will improve the CoV before committing to it; if not, the agent leaves the cluster state unchanged.

The agent reacts to failing bricks by rebuilding RPs with APs on such bricks, using spare resources it sets aside when initializing each brick. The agent uses MinDot to reconfigure any affected RPs, using spare resources if necessary. The agent may need to reconfigure RPs not directly affected by the failed bricks in order to find a feasible reconfiguration for affected RPs; it uses a backtracking algorithm to search until it finds a set of RPs with a feasible sequence of reconfigurations. The agent sets goals on affected APs of the RPs, reflecting their new configuration after all data is migrated.

The agent then repeatedly iterates through the reconfigured RPs, rebuilding or migrating one VO in one RP per iteration. In this way shrinking APs will steadily release their resources to growing APs, which in turn may allow an AP in another RP to release resources for an AP in yet a different RP. This approach contrasts with migration scheduling based on finding matchings or colorings in the demand graph [10].

Migrating and rebuilding VOs may require large amounts of IO traffic. Kybos prevents this traffic from interfering with application traffic by reserving some fraction of the performance of each brick for management actions, and by using that reserve to perform migration and rebuild IOs. If the brick is currently underutilized, its unused performance is shared fairly among all IO traffic streams.

Kybos leverages rebalancing and rebuilding to handle scheduled additions and removals of bricks in a cluster. Once the administrator attaches a brick to the cluster, the agent will consider its resources during RP rebalancing. When the administrator indicates that a brick will be removed, the agent will reconfigure RPs with APs on the affected brick, which will place a goal of eliminating those APs; VO rebalancing will migrate data to other APs.

6 Related work

The need for large, high-performance storage systems that can manage themselves and withstand failures is not new, and becomes more pressing each day. As such, several research projects and commercial products aim to fill this need. Among these, the FAB block storage project at HP Labs [7, 20] is most similar to out work. Our work in Kybos differs from FAB its emphasis on reactive self-management instead instead of on the initial modeling, design, and configuration of data [13, 14]. The CMU Self-* project [8] also uses a brick-based storage architecture, but concentrates more on protocols that withstand Byzantine failure rather than the self-management goals. Both the Farsite [1] and Oceanstore [16] projects have a peer-to-peer foundation, providing storage system functionality on top of the spare disk space available on either an entity-wide network of computers (Farsite) or a wide area network (Oceanstore).

In the product space, the Lustre File System [4] and the Panasas storage cluster [19] each provide an object-based storage and file system for use in Linux cluster environments. Lefthand Networks [17] provides similar functionality for Windows-based systems. All of these systems lack the ability to either configure themselves or maintain themselves over time in the face of system events such as failure, load change, or new resources.

7 Current status

The IO processing path now runs in a modified SAN.FS system. It stores and retrieves data on bricks via a complete Posix file system API.

The self-management components now runs in simulation. We can create RPs, and see assignments that balance load across bricks. When we simulate multiple IO streams, bricks shape the streams such that each stream gets its reserved share of performance and a fair share of any unreserved resource, with only short transient effects as a new stream starts. When a brick fails, the system quickly reconfigures affected RPs, and then rebuilds data as fast as permitted by the performance reserve held at the bricks for management actions.

The current implementation supports all of the features discussed here. The split of function between local enforcement and global decisions has worked well. The global decision algorithms are simple because they can treat bricks as a black boxes instead of having to understand their internals. If a brick accepts an AP with particular requirements, it is responsible for meeting those requirements. These results bolster our view that intelligent self-management of a storage system is possible, and that the intelligent brick architecture will lead to a robust, scalable storage system.

References

[1] A. Adya, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, and R. P. Wattenhofer. FARSITE: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment. In *Proc. of the 5th Symp. on Operating Systems Design and Implementation*, pp. 1–14. Dec. 2002.

[2] G. A. Alvarez, E. Borowsky, S. Go, T. H. Romer, R. Becker-Szendy, R. Golding, A. Merchant, M. Spasojevic, A. Veitch, and J. Wilkes. Minerva: An automated resource provisioning tool for large-scale storage systems. *ACM Trans. on Comp. Sys.*, 19(4):483–518, Nov. 2001.

[3] E. Anderson, M. Hobbs, K. Keeton, S. Spence, M. Uysal, and A. Veitch. Hippodrome: Running circles around storage administration. In *Proc. of the 1st Conf. on File and Storage Technology*, pp. 175–188. Jan. 2002.

[4] Cluster File Systems, Inc. Lustre: A scalable, high performance file system. `http://www.lustre.org/docs.html`, 2003.

[5] Thinking outside the box. *The Economist*, Sept. 2003.

[6] C. Fleiner, D. R. Kenchammana-Hosekote, O. A. Zaki, R. Garner, W. Wilcke, H. Huels, H. Lenk, M. Ries, and K. Smolin. The IBM IceCube/Lars project. Tech. Report RJ 10292, IBM Almaden Research Center and IBM Engineering & Technology Services, San Jose, CA and Mainz, Germany, May 2003.

[7] S. Frølund, A. Merchant, Y. Saito, S. Spence, and A. Veitch. FAB: Enterprise storage systems on a shoestring. In *Proc. of the 9th IEEE Workshop on Hot Topics in Operating Systems*. May 2003.

[8] G. Ganger, J. Strunk, and A. Klosterman. Self-* storage: Brick-based storage with automated administration. Tech. Report CMU-CS-03-178, Sch. of Computer Science, Carnegie Mellon University, Pittsburgh, PA, Aug. 2003.

[9] H. Gobioff, D. Nagle, and G. A. Gibson. Integrity and performance in network attached storage. In *Proc. of 2nd Intl. Symp. on High Performance Computing*, pp. 244–256, May 1999.

[10] J. Hall, J. D. Hartline, A. R. Karlin, J. Saia, and J. Wilkes. On algorithms for efficient data migration. In *Proc. of 12th Ann. Symp. on Discrete Algorithms*, pp. 620–629, 2001.

[11] IBM Corp. IBM TotalStorage Productivity Center. `http://www.ibm.com/servers/storage/software/center/index.html`, 2004.

[12] INCITS Technical Committee. Information technology - SCSI object-based storage device commands - 2 (OSD-2). `http://www.t10.org/ftp/t10/drafts/osd2/osd2r00.pdf`.

[13] K. Keeton and A. Merchant. A framework for evaluating storage system dependability. In *Proc. of DSN 2004, the Intl. Conf on Dependable Systems and Networks*, pp. 877–886, June–July 2004.

[14] K. Keeton, C. Santos, D. Beyer, J. Chase, and J. Wilkes. Designing for disasters. In *Proc. of the 3nd Conf. on File and Storage Technology*, pp. 59–72. Mar.–Apr. 2004.

[15] D. R. Kenchammana-Hosekote, R. A. Golding, C. Fleiner, and O. A. Zaki. The design and evaluation of network RAID protocols. Tech. Report RJ 10316, IBM Almaden Research Center, San Jose, CA, Mar. 2004.

[16] J. Kubiatowicz, D. Bindel, P. Eaton, Y. Chen, D. Geels, R. Gummadi, S. Rhea, W. Weimer, C. Wells, H. Weatherspoon, and B. Zhao. OceanStore: An architecture for global-state persistent storage. In *Proc. of ASPLOS IX, the Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, pp. 190–201, Nov. 2000.

[17] Lefthand Networks. Architecting a networked storage solution. `http://www.lefthandnetworks.com/library/wp.php`.

[18] J. Menon, D. A. Pease, R. Rees, L. Duyanovich, and B. Hillsberg. IBM Storage Tank—A heterogeneous scalable SAN file system. *IBM Syst. J.*, 42(2):250–267, 2003.

[19] D. Nagle, D. Serenyi, and A. Matthews. The Panasas ActiveScale storage cluster—Delivering scalable high bandwidth storage. In *Proc. of the 2004 ACM/IEEE Conf. on Supercomputing*, Nov. 2004.

[20] Y. Saito, S. Frølund, A. Veitch, A. Merchant, and S. Spence. FAB: Building distributed enterprise disk arrays from commodity components. In *Proc. of ASPLOS XI, the Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, pp. 48–58, Oct. 2004.

[21] A. S. Tanenbaum, S. J. Mullender, and R. van Renesse. Using sparse capabilities in a distributed operating system. In *Proc. of the 6th Intl. Conf. on Distributed Computing Systems*, pp. 558–563. May 1986.

[22] Y. Toyoda. A simplified algorithm for obtaining approximate solutions to zero-one programming problems. *Management Science*, 21(12):1417–1427, 1975.