StorageAgent: An Agent-based approach for dynamic resource sharing in a Storage Service Provider (SSP) Infrastructure

Sandeep Uttamchandani
IBM Research Division
Almaden Research Center
San Jose, CA 95120-6099
sandeepu@us.ibm.com

Abstract

In a SSP Infrastructure, the *resources* of the Storage Server namely cache, memory and CPU are shared in an ad-hoc manner among the clients. These resources play an important role in determining the overall Throughput and Latency of data-access. In this paper, we propose StorageAgent: A systematic, secure and efficient approach for distributing resources. Built on agent-based semantics for dynamic resource sharing, StorageAgent achieves the following goals. First, there is an efficient utilization of available resources as there are well-defined semantics for lending and reclaiming resources. Second, security of data is ensured as access to borrowed resources is controlled solely by trusted-agents. Third, fine-grain control and metering of resources used by individual clients is possible.

1. Introduction

The Storage Service Provider (SSP) market is predicted to grow to $6 billion in 2004 [6]. The fundamental parameters that characterize the Quality of Service provided by a Storage Service Provider (SSP) are overall Throughput and Latency of data-access. The value of these parameters does not solely depend on the actual disk management, but is a resultant effect of how the Storage Servers manage Memory, Cache, CPU in addition to the actual disk management. Throughout this paper, the term *resource*, refers to the combination of the following: cache, main memory and CPU cycles.

In a SSP Infrastructure, clients are statically allocated a fixed amount of disk-space. The access to this data is controlled by a Storage Server that sits in between the clients and the actual storage-devices. Since the resources of the Server have an impact on the overall Throughput and Latency of data-access, we need a well-defined and efficient mechanism to dynamically distribute these resources among the clients. In the existing SSP infrastructure, the resources are shared in an ad-hoc manner. One of the approaches is to build a throttling mechanism that prevents a single client from monopolizing all the resources. This approach is not very efficient and has the following shortcomings. First, it leads to under-utilization, as the throttling approach is conservative and withholds resources even when they are available. Second, it does not ensure security of data present in cache and memory. This is especially required if clients belong to different organizations. Third, metering of resources is not possible and the clients cannot be charged based on the actual resources used.

In this paper, we propose StorageAgent: An Agent-based architecture for dynamic resource sharing. In this architecture, the clients are allocated to a fixed amount of resources, proportional to their disk-space or I/O rate. When they need additional resources, they can borrow it from other clients, if available. The resources that are borrowed are managed and controlled by spawning Software agents. These agents have their own thread of control and keep track of the data stored in the borrowed resources. They maintain security by confining the data access. Finally, when the resources need to be reclaimed, the agents migrate to other locations.

By using agent semantics, StorageAgent represents a systematic approach for sharing resources among the clients and achieves the following goals:

- **Optimum utilization of the available resources**. As a boundary condition, it should be possible for a single client to utilize 100% of the resources if none of the other clients are using them. At the same time, there is a guarantee to reclaim these resources whenever required.
- **Security of data**. StorageAgent ensures security against disclosure and corruption of information. Any access to the borrowed resources is via agents that implement access control mechanisms.
- **Metering of resources used by individual clients**. Agents provide a fine-grained control of resources. Thus it is possible to keep a track of the actual resources used by an individual client.

The organization of the paper is as follows. Section 2 describes a few design issues of the StorageAgent architecture. Section 3 gives details of the working of agents. In Section 4, we outline the architecture. Section 5 briefly describes the related work followed by the conclusion and future work.

2. Design Issues

This section discusses the following design issues of the StorageAgent architecture:

- How are the resources partitioned and allocated to the clients.
- What performance metrics determine the need for allocating additional resources.
- Defining Agent functionality for dynamic resource sharing.

2.1. Resource Partitioning and Allocation

Each client is statically assigned a fixed amount of *resources* i.e. cache, main memory and CPU cycles (Figure 1). We refer to it as *client-share*. These are the minimum resources that the client is always eligible for and can demand them whenever required. If a particular client is not using all its resources in the client-share, the platform can temporarily allocate them to another client, with the guarantee that these resources can be reclaimed whenever required.

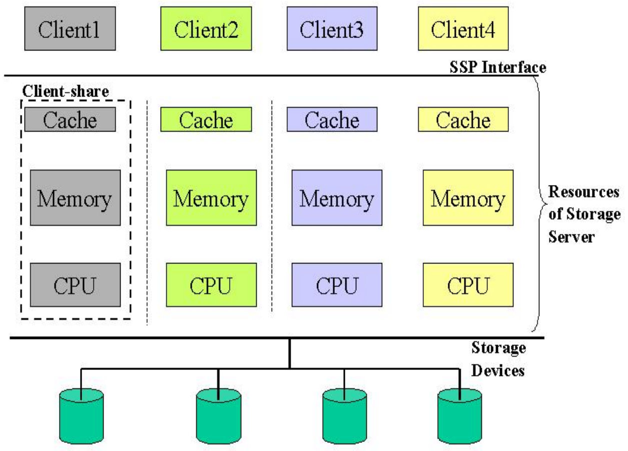

Figure 1: Partitioning resources between the SSP clients

The logical partitioning of the resources into client-shares is achieved by a Resource-Manager (RM) built on top of the operating system. The RM assigns each client-share, a fixed address-space in cache and memory. It implements access-control mechanisms to prevent clients from accessing data in other client-shares. In order to allocate CPU-cycles to each of the clients, the RM implements multiple queues, with one queue for each client. It then allocates CPU-cycles to each of these queues in a round-robin fashion.

The resources within a client-share are divided into blocks. Blocks are the smallest unit of resource allocation. Initially, when the client begins data-access, it is allocated 30% of the blocks present in its client-share. More blocks are then allocated incrementally by tracking the QoS being delivered to the client. The details of QoS-tracking are given in the following section.

When the total number of blocks being used within a client-share reaches a certain threshold value, the RM starts searching for additional resources that can be borrowed from other client-shares. If resources are available, software agents are spawned that manage these borrowed resources. In the agent terminology, the client-share that spawns these agents is referred as the home-base [7]. Whenever agents are spawned, 10% of the resources within the home-base are left un-allocated and reserved as “retract-space.” This is explained in the later sections.

2.2. Metrics for QoS tracking

To determine the need to allocate additional resources, a performance monitor is implemented for each client-share. It tracks the following system-conditions to determine the QoS being delivered:

- Number of page-faults encountered.
- Cache miss ratio.
- Checking for thrashing of data in memory.
- Number of physical to logical I/Os per transaction.
- Length of queues for I/O and CPU.

By observing these system-level conditions, the performance monitor estimates the QoS and raises trigger-events for additional data allocation.

2.3. Using Agents for dynamic resource sharing

The term Software agents has been traditionally used for pieces of logic that act autonomously on behalf of the user. They are generally divided into two categories: simple and intelligent agents. StorageAgent consists of simple agent programs, primarily designed to manage resources (Figure 2). It encapsulates a thread of control and is assigned a unique name. Logic is built into the agent to execute the following tasks:

- Track the location of data in the resources that are allocated to it.
- Provide the home-base with the physical address in response to data-access requests.
- Implement various policies for data management. For example, the agent can implement different page-replacement policies such as FIFO, LRU, etc based on data-access pattern.
- Keep a track of the available resources and advertise it to the home-base.

3. Working of Agents

This section describes the working of Agents and explains the following:

- How is the data managed by the agents and their response to data-access requests.
- How are the borrowed resources reclaimed.
- What are the mechanisms implemented for data security.

3.1. Management of Data

When the client makes a data-access request, there are four possible scenarios that can arise:

(i) Data is already present in the client-share resources.
(ii) Data is present in one of the agent-resources.
(iii) Data needs to be fetched from the disk, and there is space available in the client-share.
(iv) Data needs to be fetched from the disk and the only space available is in the agent-resources.

Cases (i) and (iii) are straightforward. Cases (ii) and (iv) involve agents and are described below.

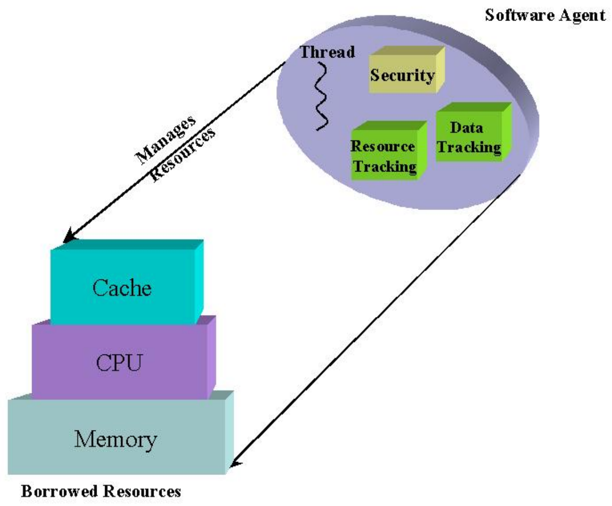

Figure 2: Software Agents manage the borrowed resources

3.1.1. Data is already present in agent-resources

To track the data maintained by the agents, each client-share maintains a resource-table. This table is similar to the one maintained by the operating system to track data in memory. Each entry consists of pair of logical address of the data and its physical location. The physical location can point to an address in memory, cache or the name of the agent. The agents maintain similar tables to track data. In case the data-entry is not found, data is fetched from the physical device.

When the resource table entry points to an agent, the home-base sends it a data-request message (Figure 3). After scanning its resource-table, the agent replies with the actual physical location of the data and a security-ticket to access it. The details of the security mechanisms are given later in the section.

3.1.2. Data is fetched from disk by the agent

When the data is not found in the resource-tables, it is fetched from the device and copied in memory. If the old data in the client-share needs to be flushed-out to make space for the new one, the resources of the agents are checked for unused space. Each agent constantly advertises the resources that are available to it. The home-base then selects the agent with maximum resources and sends the request for data-fetch. The agent fetches the data and sends the memory-address where the data is copied along with the security access ticket.

3.2. Reclaiming resources

When 80% of the resources within a client-share are used-up, it starts reclaiming the resources that have been leased to the agents. Depending upon resource availability, the agent can select one of the four options to hand-over resources:

- **Return to home-base**: For this option, the agent uses the "retract-space" that is kept unallocated in the home-base. To minimize data transfer, only the dirty pages are copied to the retract space. The read-only data is discarded and resources are handed-over. The purpose of retract-space is to ensure that the borrowed resources can be promptly returned.
- **Migrate to another client-share**: If resources are available, the agent can migrate to another client-share. It needs to update the resource tables of the home-base. The semantics of agent migration are well-studied [7].
- **Transfer data to another agent**: It can transfer the data to another agent belonging to the same home-base. In this case a forwarding address is used to redirect access requests.
- **Demise of the agent**: In case there are no resources available in other client shares and even in the retract-space of the home-base, the agent sends a high-priority interrupt to the Resource Manager (RM) to write-back all the dirty pages to disk. The resource-table entries containing the agent address are deleted.

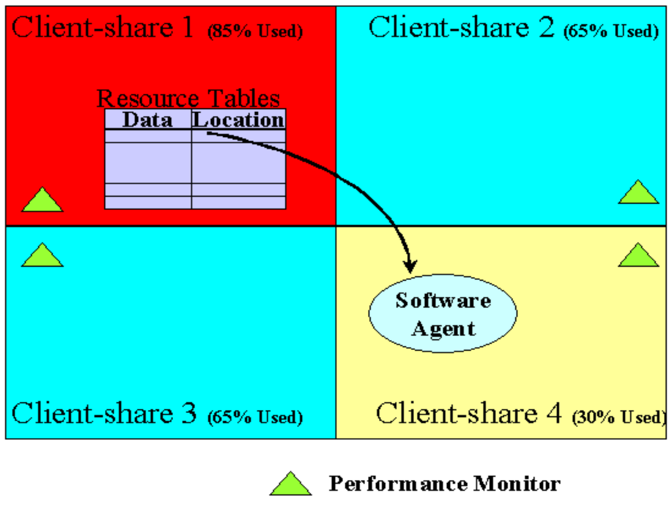
Client-share 1 (85% Used)
Client-share 2 (65% Used)
Resource Tables
Data
Location
Software Agent
Client-share 3 (65% Used)
Client-share 4 (30% Used)
Performance Monitor

Figure 3: Accessing data maintained by agents

3.3. Data Security

Security is one of the topics that is well-studied in the context of mobile agents [2, 3, 4]. The conventional issues related to agent security do not pose a threat in StorageAgent, since all the agents are spawned and maintained by the platform and the clients have no control on the actions performed by the agents or the platform on which they operate.

The fundamental security concern in StorageAgent is to prevent the client from unauthorized data-access in the resources of other clients. This is particularly challenging when resources are borrowed on other client-shares. In order to prevent disclosure and corruption of data by malicious Users, we implement the following two mechanisms:

- **Using tickets for accessing agent resources**

The home-base cannot access the agent resources without access-tickets. The agent sends these tickets in response to data-access requests. The tickets are encrypted and carry the following information: (i) The starting physical address location (ii) The allowed access-byte-range (iii) The Time To Live (TTL). The RM allows access to resources by decrypting the tickets. The TTL prevents multiple accesses by forging.

- **Data scrambling**

Before handing over the borrowed resources, the agent scrambles the data that is present in those resources. This prevents disclosure of data, after the agent has migrated.

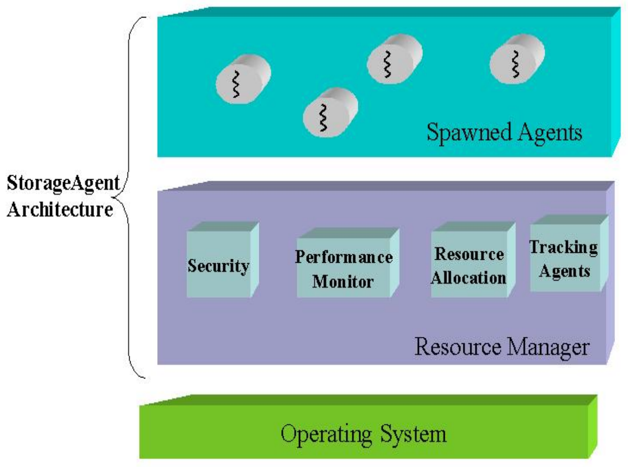

Figure 4: Components of the StorageAgent Architecture

4. Architecture

Figure 4 outlines the various components of the StorageAgent architecture, that have been mentioned in the previous sections. The architecture mainly consists of the Resource Manager built on top of the Operating system alongwith the Agents that are spawned to manage shared-resources. The Resource Manager can be implemented as a part of the Operating system [5]. This will reduce context-switches between User and Kernel space.

5. Related Work

There are numerous papers that address various issues related to the working of agents. The theoretical model of Actors [1] supports concurrent mobile objects in a programming environment. JavaSeal [2] along with [3, 4], implement mechanisms for mobile agent security. Due to space limitations, this section is not comprehensive.

6. Conclusion and Future work

This paper addresses the issue of dynamically sharing resources namely cache, memory and CPU between clients connected to a Storage Service Provider. By building an architecture based on agent semantics, we have proposed a systematic, efficient and secure approach for resource sharing. This architecture is relatively easy to implement and can utilize the current research in software agents to the benefit of Storage Management.

In the future, the StorageAgent architecture can be extended for load-management and resource-sharing among multiple Storage Servers. Further, it should be possible to manage each of the resources differently based on client-behavior. For example, each client-share can implement different page-replacement policies, etc.

7. References

[1] G. Agha, "Actors: A Model of Concurrent Computation in Distributed Systems," *Artificial Intelligence Series,* MIT Press, Cambridge, Mass., 1986.

[2] C. Bryce, and J. Vitek, "The JavaSeal Mobile Agent Kernel," *Proceedings of the First International Symposium on Agent Systems and Applications and Third International Sympo-sium on Mobile Agents (ASA/MA'99*), October 1999, pp 103-117.

[3] R. Gray, D. Kotz, G. Cybenko, and D. Rus, "D'Agents: Security in a Multiple-Language, Mobile-Agent System," In Giovanni Vigna, editor, *Mobile Agent Security,* Lecture Notes in Computer Science, Springer-Verlag, 1998.

[4] D. Hagimont, and L. Ismail, "A Protection Scheme for Mobile Agents on Java," *Proceedings of the 3rd ACM/IEEE International Conference on Mobile Computing and Networking,* September 1997.

[5] D. Johansen, R. van Renesse, and F. Schneider, "Operating system support for mobile agents," *Proceedings of the 5th. IEEE HOTOS Worksho*p, 4th-5th May, 1995.

[6] Survey of Storage Service Provider. http://www.itaa.org/isec/pubs/e20017-09.pdf

[7] PAttie Maes , Software Agent Tutorial, ACM SIGCHI Conference on Human Factors in Computing Systems,1997. http://pattie.www.media.mit.edu/people/pattie/CHI97/