Università di Pisa
Dipartimento di Informatica

Technical Report: TR-05-21

Towards a distributed scalable data service for the Grid

Marco Aldinucci Marco Danelutto Ganni Giaccherini
Massimo Torquati Marco Vanneschi

October 4, 2005

Address: via Largo B. Pontecorvo 3, 56127 Pisa, Italy. Tel: +39 050 2212700 Fax: +39 050 2212726

Towards a distributed scalable data service for the Grid*

Marco Aldinucci† Marco Danelutto‡ Gianni Giaccherini‡
Massimo Torquati‡ Marco Vanneschi‡

October 4, 2005

Abstract

Abstract: AdHoc (Adaptive Distributed Herd of Object Caches) is a Grid-enabled, fast, scalable object repository providing programmers with a general storage module. We present three different software tools based on AdHoc: A parallel cache for Apache, a DSM, a main-memory parallel file system. We also show that these tool exhibit a considerable performance and speedup both in absolute figures and w.r.t. other software tools exploiting the same features.

Keywords: Grid, Data Grid, Web caching, Apache, PVFS, DSM, Web Services.

1 Introduction

The demand for performance, propelled by both challenging scientific and industrial problems, has been steadily increasing in past decades. In addition, the growing availability of broadband networks has boosted data traffic and therefore the demand for high-performance data servers. Distributed memory Beowulf clusters and Grids are gaining more and more interest as low cost parallel architectures meeting such performance demand. This is especially true for industrial applications that require a very aggressive development and deployment time for both hardware solutions and applications, e.g. software reuse, integration and interoperability of parallel applications with the already developed standard tools.

However, these needs become increasingly difficult to be met with the growing scale of both software and hardware solutions. The Grid is a paradigmatic example. The key idea behind Grid-aware applications consists in making use of the aggregate power of distributed resources, thus benefiting from a computing power that falls far beyond the current availability threshold in a single site. However, developing applications able to exploit it is currently likely to

*This work has been supported by the Italian MIUR FIRB *Grid.it* project No. RBNE01KNFP.

†ISTI – National Research Council (CNR), Via Moruzzi 1, I-56124 Pisa, Italy

‡Department of Computer Science, University of Pisa, Largo B. Pontecorvo 3, I-56127 Pisa, Italy

be a hard task. To realize the potential, programmers must design highly concurrent applications that can execute on large-scale platforms, that cannot be assumed neither homogeneous, secure, reliable nor centrally managed. Also, these applications should be fed with large distributed collections of data.

AdHoc (Adaptive Distributed Herd of Object Caches), is a distributed *object* repository [3]. It provides applications with a distributed storage manager that virtualize Processing Elements (PEs) primary or secondary memories into an unique common memory. However, it is not just another Distributed Shared Memory (DSM), it rather implements a more basic facility. The underlying idea of AdHoc design is to provide the application (and programming environment) designer with a toolkit to solve data storage problems in the Grid framework. In particular, it provides the programmer with building blocks to set up client-server and service-oriented infrastructures which can cope with Grid difficult issues aforementioned. The semi-finished nature of AdHoc ensures high adaptability and extendibility to different scenarios, and rapid development of highly efficient storage and buffering solutions meeting industrial needs.

In this paper we discuss AdHoc and its grid-oriented features. Also, we present the design of three different AdHoc-based software tools and we compare their performance with others exploiting similar features:

1. A cache built on top of AdHoc for farms of the *Apache* Web server. It enables a farm of Apache web servers to exploit the aggregate memory space and network bandwidth of many PEs with a sensible speedup w.r.t. native Apache cache, and with no modification to the Apache core since it can be attached as plug-in.
2. A object based DSM for ASSIST [2], which is a high-level programming environment for Grid applications. AdHoc with a suitable proxy library provides ASSIST with a shared memory abstraction matching typical Grid requirements by supporting heterogeneity and dynamic availability of platforms.
3. ASTFS, a PVFS-like parallel virtual file system. Differently from PVFS1 [5], it supports heterogeneous platforms and data caching, while performing better or comparably w.r.t. PVFS working on a RAM-disk file system.

2 The AdHoc data server

The AdHoc underlying design principle consists in clearly decoupling the management of computation and storage in distributed applications. The development of a parallel/distributed application is often legitimated by the need of processing large bunches of data. Therefore data storages are required to be fast, dynamically scalable and enough reliable to survive to some hardware/software failures. Decoupling helps in providing a broad class of parallel applications with these features while achieving very good performances. AdHoc virtualizes a PE primary (or secondary) memory, and cooperating with other AdHocs, it provides a common distributed data repository.

The general AdHoc-based architecture is shown in Fig. 1. Clients may access data through different protocols which are implemented on client-side within *proxy* libraries. Proxies may act as a simple adaptors, or exploit complex behaviors also cooperating with other client-side proxies (e.g. distributed agreement, dotted lines in the figure). Both clients and servers may be dynam-

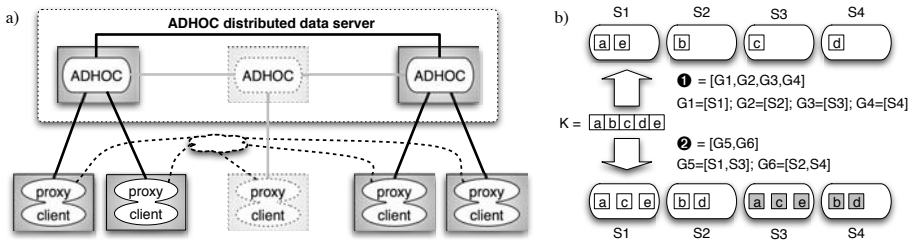

Figure 1: a) Typical architectural schema of applications based on AdHoc. b) Example of two different distribution and replication schemas (❶ and ❷) for a collection K of objects a, b, c, d over 4 AdHoc servers S1, S2, S3, S4 (grey objects are replicas).

ically attached and detached during the program run.

A set of AdHocs implements an *external* storage facility, i.e. a repository for arbitrary length, contiguous segments of data (namely *objects*). An object cannot be spread across different AdHocs, it can be rather replicated on them. Objects can be grouped in ordered *collections* of objects, which can be spread across different AdHocs.

Both objects and their collections are identified by *keys* with fixed length. In particular, the key of a collection specify to which *spread-group* and *replica-group* the collection belong. These groups logically specifies how adjacent objects in the collection are mapped and replicated across a number of logical servers. The actual matching between logical servers and AdHocs is performed at run-time through a distributed hash table. AdHoc API enables to `get/put/remove/execute` an object, and to `create/destroy` a key for a collection of objects. AdHoc does not provide collective operations to manage collections of objects (except key creation and destruction), these collective operations can be implemented within the client proxy. Each AdHoc manage an *object storage* and a write-back *cache* used to store server home objects and remote home objects respectively.

An example is shown in Fig. 1 b). Adjacent objects a, b, c, d, e of the collection K are stored in the distributed data server in two different ways (❶ and ❷). Adjacent objects of a collection are allocated and stored in a round robin way along a list of replica-groups. Each object is stored in each server appearing in the replica-group. Many spread-groups and replica-groups can be defined for a distributed data server, moreover they can be dynamically created and modified. This enables both to attach new AdHocs to a distributed server and to re-map (migrate) objects among different AdHocs within a distributed server. Once a AdHoc does not appear in any group and is empty, it can easily detached with no data loss (it can also detached at any moment, possibly with partial data loss). Object re-mapping might be an expensive operation and is supposed to be infrequent. Notice that since the collection and object keys remain unchanged in re-mapping, data may be re-mapped at run-time while keeping valid all involved keys. As an example, a distributed linked list using keys as pointers may be trasparently re-mapped.

AdHoc `execute(key,method)` operation enables the remote execution of a

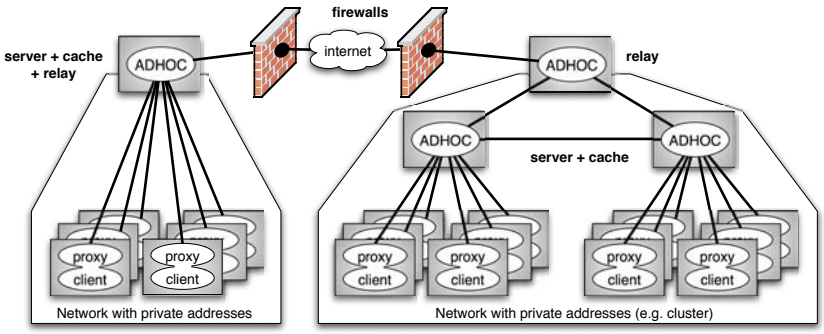

Figure 2: AdHoc-based distributed data server running onto different clusters with private address ranges and protected by firewall.

method, provided the key refers a chunk of code instead of plain data (i.e. an actual object which is executable on the target platform). This operation is meant as mechanism to extend server core functionalities for specific needs. As an example, lock/unlock, object consistency management, and atomic sequences of operations (e.g. `get&remove`) on objects have been introduced in AdHoc in this way.

As sketched in Fig. 2, AdHocs can be connected though firewalls and across networks exploiting different private address ranges. In particular:

- AdHocs can connect one another with a configurable number and range of ports. An AdHoc-based distributed server with n AdHocs can be set up across n firewalls, $n-1$ of them having outbound connectivity only, and 1 firewall having just 1 open in-bound port. However, the richer is the connectivity among servers the better is the expected performance.
- AdHocs may work as relays for others. This enable to set up a distributed data server across networks with different private address ranges, that is the usual configuration of clusters belonging to a Grid. For `get/put` objects, each connected graph of AdHocs is functionally equivalent to a complete graph. However, currently only directly connected AdHocs may belong to the same spread- or replica-group (collection cannot be spread through relays). Moreover, since AdHocs may be dynamically attached, different subgraphs are not supposed to be started all together, as may happen in the case they are executed through different job schedulers on top of different clusters.

2.1 AdHoc Implementation

An AdHoc is implemented as a C++ single thread process; it relies on non-blocking I/O to manage concurrent TCP connections [6]. The AdHoc core consists of an executor of a dynamic set of finite state machines, namely *services*, which reacts to socket-related events raised by O.S. kernel (i.e. connections become writable/readable, new connection arrivals, connection closures, etc.). In the case one service must wait on an event, it consolidates its state and yields the control to another one. The AdHoc core never blocks on I/O network operations: neither on `read()/write()` system calls nor on AdHoc

protocol primitives like remote PE memory accesses. The event triggering layer is derived from the `Poller` interface [9], and may be configured to use several POSIX connections multiplexing mechanisms, such as: `select`, `poll`, `kqueue`, and Real-Time signals. Non-blocking I/O on edge-triggered signaling mechanism is know to be the state-of-the-art of server technologies over TCP [7]. Indeed, an AdHoc can efficiently serve on a single port many clients, each of them supporting thousand of concurrent connections.

We experienced that the AdHoc-based distributed data server exhibits a close to perfect speedup in parallel configuration (many connected AdHocs), both in supplied memory room and aggregate network bandwidth. It also support heterogeneous distributed platforms, in particular it has been extensively tested on Linux (2.4.x/2.6.x) and Mac OS X (10.3.x/10.4.x). For any further detail on AdHoc implementation and testing we refer back to [3].

2.2 AdHoc as a Grid-aware software

AdHoc is a part of the ASSIST Grid-aware programming environment [1], and it is building block for Grid-aware applications and programming environments because it can cope with many of the key issues of the Grid:

- *Connectivity:* firewalls, multi-tier networks with private address ranges.
- *Performance and fault-tolerance:* data distribution, replication, and caching (parallelism and locality), dynamic data re-distribution, adaptability through dynamic reconfiguration of the set of machines composing the distributed server.
- *Heterogeneity and deployment:* it is free GPL software that can be easily ported on POSIX platforms; it has been tested on several Linux and BSD platforms; it supports heterogeneous clusters (in O.S. and CPU); it can be deployed through standard middleware (as Globus); several AdHocs composing a single distributed server do not need to start all together, thus they can be deployed on different clusters through different job schedulers.

Unlike some other approaches to data grid (e.g. European Data Grid [10]) AdHoc does not provide a rigid middleware solution for a particular problems (e.g. very large, mostly read-only scientific data). It rather provides the application developer with a configurable and extendible building block to target quite different problems, both in scientific and industrial computing, ranging from high-throughput grid data storage to low-latency high-concurrency cluster and enterprise grid data services.

3 Apache Web Caching Experiments

The AdHoc+Apache architecture is compliant to Fig. 1 a). In this case the *client* is the Apache Web server, the *proxy* is a modified version of *mod_mem_cache* Apache module and dashed lines are not present. In particular, *mod_mem_cache* has been modified by only substituting local memory allocation, read and write with AdHoc primitives.

Observe that AdHoc+Apache architecture is designed to improve Apache performance whether the performance bottleneck is memory size, typically in the case the working set does not fit the main memory. In all other cases, the

AdHoc+Apache architecture does not introduce any significant performance penalties w.r.t. the stand-alone Apache equipped with the native cache.

We measured the performance of Apaches+AdHoc architecture on the a 21 PEs RLX Blade; each PE runs Linux (kernel-2.6.x) and is equipped with an Intel P3@800MHz, 1GB RAM, a 4200rpm disk and a 100Mbit/s switched Ethernet devices. The data set is generated according to [4] by using a Zipf-like request distribution ($\alpha = 0.7$), and has a total size of 4GBytes. In all tests we used the Apache 2.0.52 Web server in the *Single-Process Multi-Threaded*. HTTP requests are issued by means of the *httperf* program. In Fig. 3 we compare Apache against Apache+AdHoc performances. The test takes in to account three basic configurations:

① an Apache+AdHoc running on different PEs, AdHoc exploiting 900MB of *object storage* total memory accessed by all Apache threads.
② a stand-alone Apache with no cache.
③ a stand-alone Apache with the *mod_mem_cache* (Apache native cache) exploiting a maximum of 900MB.

As shown by ③, the Apache with the original cache lose its stability when the requests rate grows. In this case, Apache spawn more and more threads to serve the increasing pressure of requests, inducing harmful memory usage: the competition of cache subsystem and the O.S. in both memory space and memory allocation leads the O.S. to the swap border resulting in a huge increase of reply latency. Quite surprisingly the Apache with no cache performs even better (②). In reality this behavior is due to the File System buffer that acts as a cache for Apache disk accesses, and which gracefully decrease its size in case the system requires more memory to manage many threads avoiding swapping. In this case the performance also depends on site organization on disks. In general FS cache is unsuitable for Web objects since requests do not exploit spatial and temporal locality w.r.t. disk-blocks [4]. Moreover, FS cache is totally useless for dynamic Web pages, for which we experienced the effectiveness of the Apache native cache module [8]. As a matter of fact, the 2PEs figures (③) confirm that mapping Apache and AdHoc on different PEs significantly improves performances.

As shown in [3], the gain of the Apache+AdHoc architecture is even more greater for Apache Multi-Process Multi-Threaded configuration since Apache processes can share a common memory through AdHoc. Additional experiments on parallel configuration confirm that a single AdHoc may support many Apaches with a very good scalability [3].

4 An AdHoc-based DSM for ASSIST

ASSIST is a programming environment aimed at the development of distributed high-performance applications on Grids [2]. ASSIST applications are described by means of a coordination language, which can express arbitrary graphs of either sequential or parallel components, which may exploit distributed shared data structures.

Up to now, these data structures was stored in a standard DSM implemented within ASSIST run-time. This DSM was implemented as a library providing typed global pointers (called *references*), it requires the full connectivity with all parterns, and can hardly deal with firewalls. The last version of ASSIST (v1.3) includes a novel AdHoc-based implementation of references. This new

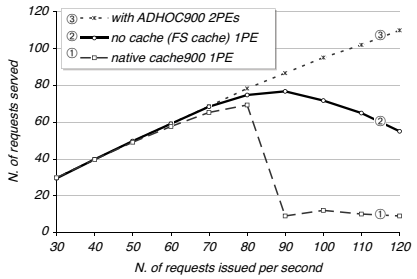

Figure 3: Evaluation of AdHoc as Apache cache.

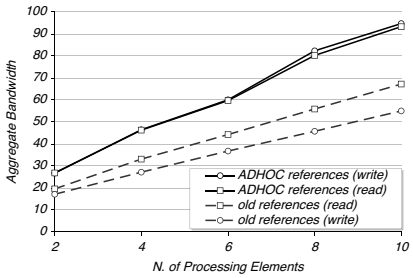

Figure 4: Evaluation of AdHoc-based DSM.

implementation overcome mentioned problems of the previous version while improving the overall DSM flexibility and performance:

- Data is stored externally to computation in specialized servers. The data can now survive to application lifespan, and it can independently mapped from computational activities. Figure. 4 describes aggregate bandwidth of reading/writing a spread array (`int[800M]`, as a collection of 16KB objects). The AdHoc-based solution exhibit a close to optimal absolute figures, and a clear performance gain with respect the previous version of the DSM.
- AdHoc can be dinamically started and stopped to meet variable application requirements and Grid platforms avalaibility over time.
- Data can be migrated among different AdHocs to implement dynamic load-balancing schemas for memory allocation and network throughput, meeting Grid platforms unsteadiness and varing load over time.
- AdHoc-based distributed data server can easily wrapped to supply a HTPP/SOAP data service for the Grid.

5 An AdHoc-based Parallel File System

The Parallel Virtual File System (PVFS) [5, 11] is one of the most used high-performance and scalable parallel file system for PC clusters that requires no special hardware.

In order to provide high-performance access to data stored on the file system by many clients, PVFS spreads data out across multiple cluster I/O nodes (IONs). By spreading data across multiple I/O nodes, applications have multiple paths to data through the network and multiple disks on which data is stored. This eliminates single bottlenecks in the I/O path and thus increases the total potential bandwidth for multiple clients, or aggregate bandwidth. Metadata stored in a special node, called manager (MNG). Metadata is information which describes a file, such as its name, its place in the directory hierarchy, its owner, and how it is distributed across nodes in the system. When reading a file from PVFS, a client contacts MNG node to retrieve file meta-data, then it gather file parts, each from the proper IONs. As show in Fig. 5 a), a PVFS client may benefit from the concurrent connection to many IONs, thus benefiting from network aggregate bandwidth.

The AdHoc-based FS (called ASTFS) implements the same functionalities of PVFS, and exploits a similar API. ASTFS API is realized by means of a proxy library linked to the application client (see Fig. 1) which tranlates file-oriented commands in a sequence of AdHoc commands. Figure 5 c) reports a comparison, in terms of aggregate bandwidth at the client ends, between PVFS and ASTFS (tested architectures are shown in Fig. 5 a) and b), respectively); ASTFS is configured to exploit the full connectivity among AdHoc servers. As shown in Fig. 5 c), PVFS and ASTFS (working on RAM-disk) exhibit quite the same bandwidth, which should be considered a very good result since PVFS represent the state of the art of distributed FS for clusters. ASTFS, differently from PVFS, neither requires the full connectivity among PEs (client-server, server-server). As a matter of fact, grid nodes does not exhibit a complete graph of links/connections due to the multi-tier network structure, firewalls, private address ranges in clusters, etc. We also experienced that a single link between client is not a limiting factor for client throughput towards the servers. In particular, ASTFS multi-threaded proxy succeeds to pipeline multiple requests on a single link relying on the ability of AdHoc in sustaining a very high number of concurrent requests on it.

ASTFS has been primary designed to manage main memory allocation, but it can be easily configured to use disk as storage. It also exhibits some additional features with respect to PVFS, such as the support for heterogeneous platforms, and for caching of read-only opened files. As shown in Fig. 5 c) the FS performance significantly increase whether each AdHoc is configured to exploit a little cache (10 MB). As typical for caching, the equivalent speedup in terms of aggregate bandwidth is super-linearly boosted.

The same test has been also performed in the Fig. 2 scenario: the two 6PEs clusters are hosted in two different institutes of Pisa Uni. (each of them protected by its own firewall with one open port), they are internally linked with a fast Ethernet, and connected one to the other with a 2 MB/s link (average). Just the front-end machine of each cluster can connect to its counterpart in the other cluster. On this scenario we experienced an aggregate bandwith at clients ends of 34 MB/s with no cache. Notice, that the AdHoc hosted in the font-end node can be configured to exploit a cache of remote sites data considerably improving the overall performance.

6 Conclusions

Overall, we envision a complex application made up of decoupled components, each delivering a very specific service. Actually, AdHoc provides the programmer with a data sharing service. We introduced AdHoc, a fast and scalable "storage component" which cope with many of Grid key issues, such as performance, adaptivity, fault-tolerance, multi-site deployment and run (despite of firewalls, job schedulers, private addresses). We shown that AdHoc simplicity is its strength: it enabled the rapid design and development of three different memory management tools in few months. These tools exhibit a comparable or better performance with respect their specialized counterparts. In some cases applications relying on these tools became ready for the Grid with no modifications to their code. AdHoc is freely available under GPL license as part of the ASSIST toolkit.

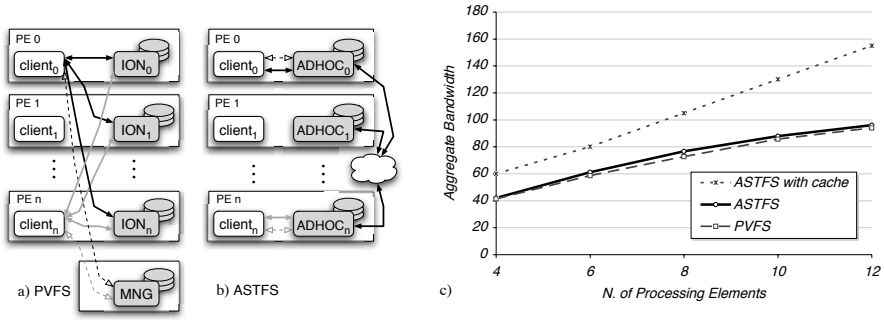

Figure 5: a) Many clients accessing a PVFS. Each PE hosts a client is required to be a ION. Solid edges show data paths, dashed edges meta-data paths. b) Many clients accessing a ASTFS. c) Comparison among PVFS and ASTFS. Each client read a partition of the file in segments of 10MB randomly chosen within its partition. In both cases, each client reads the same 800MByte file spread among all PEs.

References

[1] M. Aldinucci, S. Campa, M. Coppola, M. Danelutto, D. Laforenza, D. Puppin, L. Scarponi, M. Vanneschi, and C. Zoccolo. Components for high performance Grid programming in Grid.it. In *Proc. of the Workshop on Component Models and Systems for Grid Applications*. Springer, January 2005.

[2] M. Aldinucci, M. Coppola, M. Danelutto, M. Vanneschi, and C. Zoccolo. ASSIST as a research framework for high-performance Grid programming environments. In J. C. Cunha and O. F. Rana, editors, *Grid Computing: Software environments and Tools*. Springer, 2005. To appear.

[3] M. Aldinucci and M. Torquati. Accelerating apache farms through ad-HOC distributed scalable object repository. In M. Danelutto, M. Vanneschi, and D. Laforenza, editors, *10th Intl Euro-Par 2004: Parallel and Distributed Computing*, volume 3149 of *LNCS*, pages 596–605, Pisa, Italy, August 2004. Springer.

[4] L. Brelau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and Zipf-like distributions: Evidence and implications. In *Proc. of the Infocom Conference*, 1999.

[5] P. H. Carns, W. B. Ligon III, R. B. Ross, and R. Thakur. PVFS: A parallel file system for linux clusters. In *Proc. of the 4th Linux Showcase and Conference*, pages 317–327, Atlanta, GA, USA, October 2000.

[6] A. Chandra and D. Mosberger. Scalability of Linux event-dispatch mechanisms. Technical Report HPL-2000-174, HP Labs., Palo Alto, USA, December 2000.

[7] L. Gammo, T. Brecht, A. Shukla, and D. Pariag. Comparing and evaluating epoll, select, and poll event mechanisms. In *Proc. of the Ottawa Linux Symposium*, Ottawa, Canada, 2004.

[8] A. Iyengar and J. Challenger. Improving Web server performance by caching dynamic data. In *Proc. of the USENIX Symp. on Internet Technologies and Systems Proceedings*, Berkeley, CA, USA, Dec. 1997.

[9] D. Kegel. *Poller Interface*, 2003. http://www.kegel.com/poller/.

[10] E. Laure, H. Stockinger, and K. Stockinger. Performance engineering in data Grids. *Concurrency and Computation: Practice and Experience*, 17(2–4):171–191, 2005.

[11] *The PVFS home page*. http://www.parl.clemson.edu/pvfs/index.html.