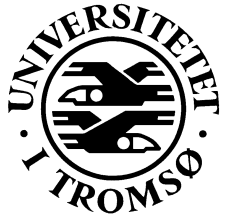

REPORT

Computer Science Technical Report: **94-17** **September 1994**

Increased Scalability of the StormCast Distributed Application

V. Farsi

INSTITUTE OF MATHEMATICAL AND PHYSICAL SCIENCES
Department of Computer Science
University of Tromsø, N-9037 Tromsø, Norway, Telephone +47 77 64 40 41, Telefax +47 77 64 45 80

Increased Scalability of the StormCast Distributed Application

V. Farsi

Department of Computer Science,
Institute of Mathematical Sciences,
University of Tromsø,
N-9037 Tromsø, Norway

Abstract

Scaling is recognized as a primary concern in developing distributed applications. Scaling problems occur when an application reach its upper boundary in some way or another. This paper describes the considerations made in the redesign of the StormCast distributed application to achieve more scalability. Naturally, there can be many factors that restrict a system's scalability. The challenge meeting the designers of distributed applications is to limit the influence of such factors. In this paper we focus on modularity, decentralized approach to design of application, transparent naming, caching of information, replicated data and services in order to enhance scalability.

1 Introduction

Scalability is one of the properties which distinguish distributed computing radically from both centralized and network computing. Unlike centralized time sharing environments, ideally, there is no limitation for the maximum size of a distributed computing environment. Naturally, there can be many factors that restrict a system's scalability (Mullender, 1989). The challenge meeting the designers of distributed software is to limit the influence of such factors.

The scaling of a distributed system or application reflects the successfulness of the system. As a distributed system or application grows, enabling more users (or groups of users) to communicate and more information to be shared, it becomes an increasingly valuable resource. There is then a considerable incentive to allow users who were originally outside the scope of the system to be included in it (Satyanarayanan, 1992).

Distributed systems and application can grow easily by using additional components. This modular growth has an important feature for scaling. Adding new components (processors or processing nodes) makes it possible to decompose applications into threads of control that execute in parallel. This kind of parallelism is often used to enhanced higher performance. Specially in distributed environments based on a workstation model, adding new users implies adding new processing nodes (like a workstation). These will all be part of the common resources. In other words the benefits of incremental growth will be much higher than the disadvantages.

The objective of the StormCast project is to construct distributed applications in a controlled manner. One of the fundamental questions in this respect is the degree of scaling of the system. This paper addresses scaling of distributed applications and presents necessary actions to be taken in order to increase the scalability of the StormCast application. The discussion covers the effect of modularity; naming and scalability; replication and caching; interprocess communication; and, administration and heterogeneity.

2 Overview of the StormCast version 2.1 application

The StormCast application has been developed in several iterations (Hartvigsen and Johansen, 1988; Johansen and Hartvigsen, 1991; Johansen, 1993). The architecture of the StormCast version 2.1 is illustrated in figure 1. The architecture consists of six layers: the monitoring layer, the collection layer, the data storage layer, the application layer, the information storage layer, and the user interface layer.

The monitoring layer consists of a set of loggers and monitoring stations. A logger is a dedicated computer which is connected to a few sensors. The logger converts analog input from the sensors to digital values and calibrates the data to physical quantities. The logger also stamps the data with the current location and time values. Usually, a logger requests data at certain predefined intervals and store them in a local cache. The local caching is done in order to be able to reply to the requests even if the data can not be obtained from the sensors.

The collection layer is responsible for collecting the monitored data from the bottom layer. The data are filtered and transmitted to the upper layer. The collection layer consists of a set of replicated collection modules. Each replicated module is responsible for the collection of raw data from one or several geographical areas. Higher layers can request data from a certain area. These requests are handled by the collection layer. For the reason of availability, the requested data will also be cached locally in this layer. If current data is impossible to obtain from the monitoring layer, cached data will be returned as responses on requests from higher layers. The timestamps always follows the data, so the higher layers can validate the data based on these timestamps.

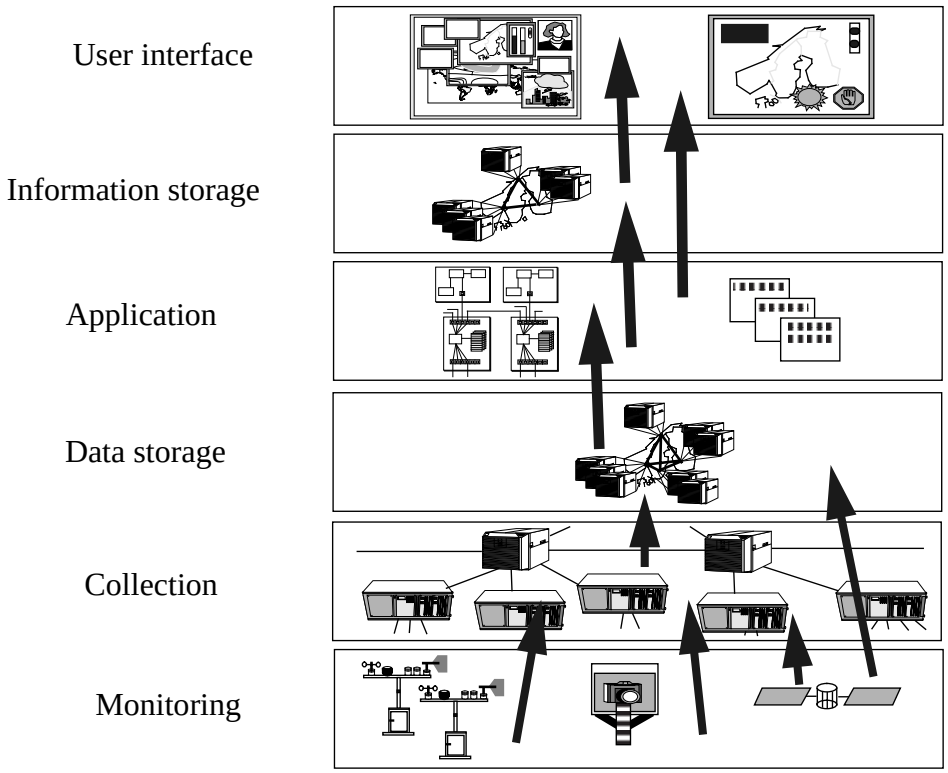

FIGURE 1. The StormCast architecture.

The data storage layer is the first layer where data is permanently stored. This layer consists of a set of replicated modules too. These modules send request to the collection layer modules concerning data from certain geographical areas at predefined frequencies. These modules store the obtained data and replies with retrieved data when requested from higher levels.

The application layer contains different types of services (applications) converting raw data into information. The modules in this layer request data from collection layer, storage layer or other applications.

In the StormCast application a distinction is made between the raw data and the information (processed data). The information storage layer provides the same set of services as the data storage layer, but information is stored at this layer.

The user interface layer creates and manages the user dialogues. The layer has mainly two tasks. Collection of data and information requested by the user, and presentation of the collected data or information to the user. Typically, a client module is activated for each requested service. This client module interacts with one or several services, logically found

in lower layers of the architecture to get the information requested. The presentation of data is done by using weather maps and other visualization techniques.

3 Scaling aspects of StormCast

3.1 Modularity

High modularity is one of the features of the StormCast architecture. A modular and layered architecture is a recognized way to structure modern systems and applications. A modular and layered architecture enhances the scalability of the application because such an architecture eases the changes in the application or system and presents the different levels of abstraction. Software engineering recognizes the maintenance and modification as an important activity in the software life-cycle. A module is used exclusively through an external interface. The internal structure and data in the modules are encapsulated. Such an encapsulation minimizes the changes in environments as a result of change in a module.

The StormCast application has gone through a lot of changes to improve the scalability. One of the latest changes in StormCast was motivated by enhancing modularity and decentralism. As described earlier, the collection, the data storage and the information storage layers consist of modules which collect data or information from layers below and serve the requests from layers above them. In the earlier versions of StormCast (up to version 1.9) these two functionalities were implemented in the same process. The problem with this approach was that these modules could easily be overloaded by requests from higher level clients, while they were blocking and waiting for reply from lower layer services. Using light weight processes in StormCast version 2.0 and 2.1 reduced but not solved this problem. Waiting for reply from other servers and replying to requests in the same time made these processes potential bottlenecks, especially when data should be collected from different lower layer servers.

In order to reduce this potential bottleneck, Farsi (1993) have separated these to functions using to different modules, a collector module and a serving module. The collector module sends request to the lower layer servers, receives data or information, and hands them to the serving modules. The serving module will just receive data and requests from other modules.

3.2 Naming

One of the fundamental issues in any application is how different entities are named. Names have been widely used in centralized computer systems as a convenient way of referring to shared resources and entities. In a distributed environment there are more resources and entities to name and the amount of resources are growing. Furthermore these entities are spread over (a) network(s) connecting all processing and peripheral devices. A naming facility for a distributed system should thus be involved not only in different entities but also in communication (Goscinski, 1991).

Naming is closely related to the location of the shared entities. In a distributed environment there are three basic terms related to locating an entity: a name, an address, and route. The name of an entity specifies what a user or a process seeks. An address specifies where this entity is, and the route specifies how to get there. For the users of a distributed application it is inconvenient to remember the address and the route. This is specially the fact when the application grows in size. For users of a large application it is convenient to remember a logical name. Names used with such a location transparency has other benefits too. In a distributed environment it is often necessary or desirable to move entities like processes and files from one node to another. The use of location transparent names allows such movement without forcing the users to learn about the new location of entities.

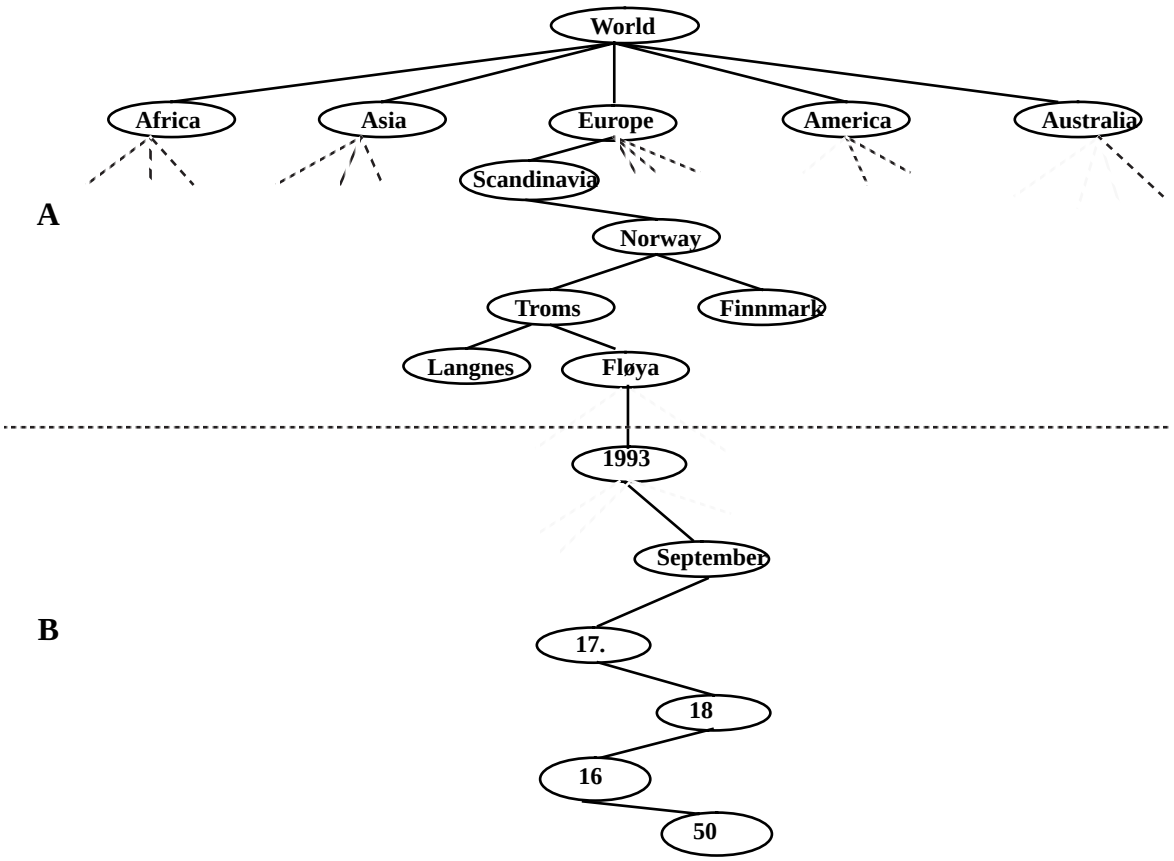

FIGURE 2. A hierarchial name space in StormCast.

In the StormCast application we have used a transparent name scheme. Each kind of data entities constitutes a hierarchial structured name space. As illustrated in figure 2 the logical name for a data entity is a leaf node in the hierarchial structured name space for that kind of

data entities. Such a partitioned logical name consists of information about the location the data was collected and the size of the collected data. Users of the application can use this logical name to identify any kind of collected data in a transparent way. The actual location of the data is detected by an interface process which contacts the name service and obtains the address of the server or service group holding that data and the file in which the data is stored. The information are handed to the processes which collect the data or information from lower layers. Both address, route (i.e., path name) and file name are transparent to the users.

The name service of StormCast is decomposed in different services located in the different layers of the application. This is done to enhance decentralism and hence scalability. The generation of name for a data entity is done by the server or service which keeps that entity. The name services are only responsible for storing name and location information from the services in their layer and responding to the location information from the clients in upper layers. In other words, the activities related to generating and retrieving of names are decentralized to avoid scaling problems.

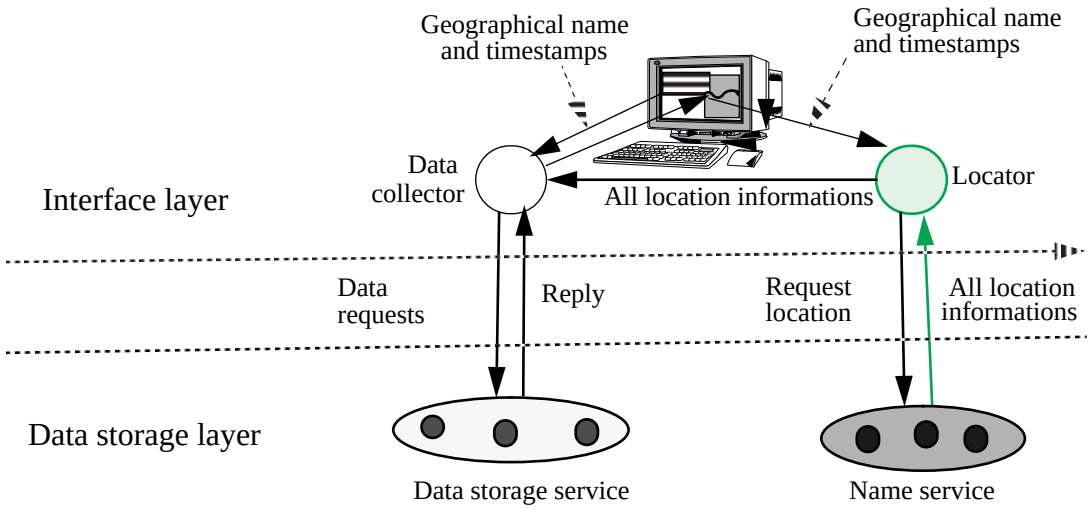

FIGURE 3. The interactions for locating and collecting data.

4 Data storage and access

Data and information are permanently stored in the data storage and the information storage layers respectively. All stored data and information in StormCast are immutable – once data or information is stored permanently it will not be changed. This property eases access to the stored data. The main kind of data stored are logger data. We have chosen to store this kind of data in sequential files. This is done to avoid the centralized approach a traditional data base represents. Logger data is stored on files sequentially and retrieved by using an index

and the number of bytes. The index and the number of bytes are easily computed knowing the time stamp of the first stored data-unit, the timestamps for period of time data is wanted, the length of a data-unit in bytes and the interval by which data is stored on file. This computation is done by the client process that requests the data. One way to enhance scalability is to place as much work as possible on clients. A client will receive the time stamp of the first stored data-unit in a file as a part of the location information from the name service. Other necessary information for this computation, like the storing interval and the length of a data-unit, are constant.

The use of files is also motivated by the caching of whole files at the client-node. This will be discussed in the next chapter.

Replication of stored data in StormCast is a way to enhance availability of data and hence scalability. Data replication in Stormcast is based on replicated storage services. Replicated services represent a decentralized approach to the accessing of data. This kind of replication makes it possible to send the different requests to different servers, or choose the nearest server, which again increase the performance.

4.1 Caching in StormCast

Caching of data and information is motivated from performance, operability and availability aspects. These are all critical to scalability. Some distributed operating and file systems like NFS (Sandberg et al. 1985), AFS (Satyanarayanan, 1990), Cedar (Satyanarayanan et al., 1990), Amoeba (Mullender, 1991), and Cedar (Schroeder et al., 1985) cache the entire files. Such a caching enhances scalability by reducing server load and network traffic.

To improve performance and availability in StormCast, data and information are cached in main memory (warm cache) by servers and clients. Caching of whole files on the clients local disk (cold cache) is an other key to improved scalability in StormCast. Whole file caching is specially suitable in StormCast because of the immutable data. In order to maintain cache coherence, propagating modifications are major considerations in any caching. The immutable data (files) in StormCast eliminates both of these problems. In other words a cached file is a read only file for the client, meaning that no cached file will be modified by the client. A server, on the other hand, can only add to a file, meaning that the data in a cached file is always valid.

We have implemented two mechanisms for whole-file caching. The first one is a procedure (stub) attached to the client process which asks for data. The other mechanism is an independent process which interacts with both clients and servers by message passing. As expected, our tests showed that the mechanism implemented as a procedure gives much better performance compared to the other one. The caching mechanism developed as an independent process has, however, another advantage. One of the side effects of scaling is inclusion of different computers, some of these computers do not have enough disk space to cache the entire files. Such a caching mechanism may serve several nodes in a cluster. We believe that the latency in a wide area network makes such a mechanism attractive.

We have used a least-recently-used algorithm (LRU) to minimize the disk space used for caching. Whenever the size of disk space used for caching is going to exceed the limitation, the least recently used files are replaced by new files.

5 Future works

This paper has addressed scalability aspects considered in the StormCast project. In the following we briefly discuss additional aspects of scalability which will be revisited due to further growth of the application.

5.1 Security

The growth and the geographical spreading of a distributed application increases the anonymity of the users (Satyanarayanan, 1988). Enforcing security in large distributed applications is a more critical task. It consists of three activities: user authentication, resource protection and communication security. The significance of authentication is that in a distributed application like StormCast it is specially important to make users accountable and responsible to the application as a whole and not only to the local UNIX system. The resource protection should consider the privileges a user has in using the resources in the application. Again, using the local UNIX rights can not cover this purpose, because the accountability in the local UNIX system differs from accountability to the widespread application which includes many local systems.

5.2 Heterogeneity

The StormCast application is developed in a homogeneous environment. The application is, however, going to be used as a workbench at different meteorological institutes. These offices are heterogeneously equipped. A widespread use of StormCast depends on the adaptability to different platforms. There are different ways to meet heterogeneity; portable RPC package and customized mechanisms are methods widely used to avoid heterogeneity problems. Communication in StormCast is mainly an extended RPC, making it possible to check and convert the RPC-packages wherever necessary. On the other hand we have already developed a mechanism for caching of files which lets a computer act as a surrogate for a cluster of other computers. We believe that such mechanisms combined with portable-RPC packages will meet the heterogeneity problems in StormCast.

5.3 Administration

Scaling implies assigning a distributed application to a wide range of centralized systems. Such an application is unlikely to be managed as a monolithic entity. Administration illustrates a difficulty related to any distributed software, namely the combination of global control with autonomy in the underlying systems. Management in StormCast makes it necessary to delegate the responsibility into the institutional boundaries, defined by geographical limitations. However, such a delegation should be linked with the authorization

(i.e., giving privileges) to some global system administrators. Another related aspect is to produce some special administrative interfaces for every software and hardware entity dedicated to the application (Schroeder, 1993). Any component can be managed by calling its administrative interface. Such an interface is dedicated to administrative tasks and is not used for other purposes. A common way is to automate the administrative tasks using RPC-calls to the interfaces by administrative programs.

6 Conclusion

In this paper we have, based on experiences with the StormCast application, discussed some central issues regarding the growth of a distributed application. A modular, layered and decentralized architecture makes a good platform for scalability of an application. Modularity and decentralism reduce the complicity of a large application by offering different levels of abstraction and ease changes which are unavoidable as an application grows.

Other scalability aspects in the design of StormCast have been transparent naming and name service generating and administrating names in a decentralized. Non-transparent names force the users to remember the name of different nodes and is thus difficult to use in large environments. The StormCast application presents the users with logical names for data and information which do not refer to the physical placement of data.

Replication and widely use of caching is another way to enhance the scalability. In StormCast we have developed mechanisms to cache whole files in the client sites. Whole file caching reduces the traffic in a network and increases the availability of data. Increased network traffic and availability are critical aspects when an application grows.

Security, heterogeneity and administration of a large application are critical aspects regarding to further scaling of the StormCast application. These will all have to be addressed in future.

References

Goscinski, A. (1991). “Distributed Operating Systems, The logical Design”, Addison-Wesley.

Hartvigsen, G., Johansen, D. 1988. StormCast – a Distributed Artificial Intelligence Application for Severe Storm Forecasting, I: M.G. Rodd, T.L. d’Epinay (Red.), *Distributed Computer Control Systems* 1988. Proceedings of the Eight IFAC Workshop (Vitznau, Switzerland, 13-15 September, 1988). Oxford: Pergamon Press, 1989. s. 99-102.

Johansen, D., Hartvigsen, G. (1991). “StormCast - a distributed application”, EurOpen spring 91, Conference Proceedings.

Johansen, D. (1993). “StormCast - A Computing the Future Approach”, In: “The StormCast Applied Approach to Distributed Computing”, Department of Computer Science, University of Tromsø, March 1993.

Mullender, S.J. (1989). “Distributed Systems”, chapter 1, Addison-Wesley publishing company, 1989. ed. Sape J. Mullender.

Mullender, S.J. (1991). “Experiences with Amoeba”, EurOpen spring 91, Conference Proceedings. Tromsø 20-24 May 1991.

Needham, R.M. (1989). “Names”, in: “Distributed Systems”, Addison-Wesley publishing company, 1989. Ed.Sape J. Mullender.

Sandberg, R., Goldberg, D., Kleiman, S., Walsh, D., Lyon, B. (1985). “Design and Implementation of the Sun network file system,” Proc. Usenix Conf. Portland, 1985.

Schroeder, M.D. (1993). “A State-of-the-Art Distributed System: Computing with BOB”, in: “Distributed Systems”, ed. S. J. Mullender, Addison-Wesley, 1993.

Schroeder, M.D., Gifford, D.K., Needham, R.M. (1985). “A caching file system for a programmer’s workstation.”, in Proc. 10. Symp. on Operating System Principles, Dec. 1985.

Satyanarayanan, M. (1988). “On the Influence of Scale in a Distributed System”, 10th International Conference on Software Engineering April 11-15, 1988 Singapore.

Satyanarayanan, M. (1990). “Scalable, Secure, and Highly Available Distributed File Access.”, IEEE Computer 23(5). May 1990.

Satyanarayanan, M. (1992). “The Influence of Scale On a Distributed File System Design”, IEEE Transactions On Software Engineering 18(1) Jan. 92.

Satyanarayanan, M., Kistler, J.J., Kumar, P., Okasaki, M.E., Siegel, E.H., Steere, D.C. (1990). “Coda a Highly Available File System for a Distributed Workstation Environment”, IEEE Transaction on Computers 39(4), April, 1990