JSEB (Java Scalable sErvices Builder): Scalable Systems for Clusters of Workstations

Maria Barra, Giuseppe Cattaneo, Umberto Ferraro Petrillo, Vittorio Scarano
Dip. di Informatica ed Appl. "R.M. Capocelli" Universita' di Salerno
84081 Baronissi (SA) – Italy

Abstract

We present a report on JSEB (Java Scalable Service Builder) whose goal is to offer programmers a tool that can be used to efficiently add scalability and fault-tolerance to a replicated service in cluster(s) of workstations.

1. Introduction

During last years we have witnessed an impressive amount of interest in cluster computing. Besides efficiency, availability and economicity, one of the reasons of this interest is the need to build scalable services that are able to handle a large number of users requests. In this situation, one of the most effective solutions is to replicate the service using a cluster of inexpensive and widely available small workstations (i.e. PCs) instead than having a single server machine. The choice of replicating a service implies several problems such as the incremental scalability, the load balancing and the fault trasparency. These problems can be addressed by making an optimal distribution of the clients' requests to the cluster of servers and by adopting strategies for fault recovery.

Offering scalable services is particularly critical nowadays because of the pressure and the strain that the World Wide Web induces on any distributed service. Several impressive real examples involved the CNN network web site after the publishing of Starr Report, the US Geological web site after a earthquake in the Bay area in San Francisco and the Microsoft ftp site when new software is released.

JSEB provides a support tool for building client-side scalable services. JSEB model is based upon a communication and operational framework used to monitor the state of several clusters of servers offering a replicated service. A JSEB client can attach itself to the monitoring framework to obtain the servers state information report. According to information obtained by monitors each JSEB client is able to pick the best available server for a given service.

JSEB offers a general solution to provide scalable services on cluster of PCs indipendently from the specific protocol. The scenario we are planning is the following (few examples are provided at the end of the paper): one has a "traditional" client-server system and wants to make it highly available by providing replicated servers on a cluster of PCs. Then, by "attaching" to each server a Server Monitor (details will be provided later) and designing the client "around" a Client Monitor, one has (for free) a hierarchical infrastructure that is able to collect information about servers' state and "provide" it to the clients that can, therefore, direct their queries to the "best" server.

JSEB is *general* since it offers an infrastructure for any protocol and for any load metric the designer can choose. It is *efficient*, since it tries to minimize the communication load among servers and clients, it adds *scalability* to an existing client-server architecture, since it allows load balancing among the servers in the cluster, it is *fault-tolerant*, since it is able to detect faults and consequently direct clients toward functioning servers.

The reader would notice that JSEB architecture can be easily extended client-side by integrating proxies for a more efficient communication with clients. Since our paper is tailored to present JSEB basic characteristics with a particular emphasis on the server side, we choose not to present such extension but it must be considered, in the rest of the paper, that every reference to *clients* can be substituted by *client or proxy*.

2. Related work

In order to identify JSEB characteristics, we present here several important parameters that must be taken into account for a scalable service.

• *Servers CPU load:* How much the servers' CPU load is affected by the overhead needed to scale the service.
• *Clients CPU load:* How much the clients' CPU load is affected by the overhead needed to scale the service.
• *Server-to-Server communication load:* How much is the traffic on the server network communication links needed to scale the service.
• *Client-to-Server communication load:* How much is the traffic on the client-to-server communication links needed to scale the service.
• *Servers Network topology:* The connection topology and the connections bandwidth of the servers' replicating a service.
• *Servers-Clients ratio:* How many servers with respect to the number of existing clients.
• *Server Replication trade-off:* How much overhead is required to replicate a service through a cluster of servers.

The scalability issue can be solved both with general-purpose solutions and special-purpose solutions. General-purpose solutions can be applied to a wide range of services but are less effective and more difficult to characterize than special-purpose one.

Among the general-purpose solutions there is DNS aliasing: each server of the pool can be reached using the same Internet literal address. Each time a client needs to make a service request to one of the servers, it makes a resolution query to the DNS system to resolve the unique server address. The DNS system will return the real address of one of the server machines in the cluster choosed using a round-robin algorithm. Since the DNS aliasing technique relies on the queries made to the DNS systems by the service clients, it trickles down the scalability problem to the DNS service, that, now, becomes heavily loaded and, possibly, a bottleneck to the overall performances since it has to manage all the resolution requests coming from the clients. This solution does not affect the *servers CPU load* while it requires a sligth amount of *clients CPU load* for the DNS queries.

A different general-purpose solution is presented in [Hunt98] and in [Anderson96]. This kind of solution is working at TCP level and service requests by the clients are trasparently routed to one of the servers by a device, directly connected to the network, that forwards each client request to a server using a round robin distribution algorithm. This approach to the scalability problem does not affect the *servers CPU load* neither the *clients CPU load* .

Smart Clients [Yoshikawa97] propose a client-side approach to provide transparent access to scalable services. Smart Client API offer two service-specific Java applet. The *client interface* applet provides the interface to the user and makes the request of service. The *Director* applet makes server requests to the appropriate (least loaded) server, and updates its notion of server state. The Smart Clients solution requires a fair amount of *servers CPU load* and *Server-to-Server communication load* while requiring a consistent amount of *clients CPU load* and *Client-to-Server communication load* because the state update information and server selection functions are client oriented.

Since special-purpose solutions can be tailored on the needs of a specifical service, we can distinguish several solutions according to the service we are scaling.

• **HTTP:** First of all, let us consider HyperText Transfer Protocol (HTTP). NCSA [Katz94] proposed a solution in 1994 that is based on the DNS aliasing technique. The NCSA approach builds a multi-workstation HTTP server using a cluster of workstations, the service requests to these workstations are assigned using a DNS aliasing system.

Another solution proposed for HTTP is SWEB [Andresen96] that requires the cluster of servers to be connected through a fast LAN. Scalability is achieved by actively monitoring the run-time CPU, disk I/O, and network load of system resource units. Each time an HTTP request is made to a server, it is parsed and then dynamically redirect to propers nodes for efficient processing. This approach requires a consistent amount of *servers CPU load* and *Server-to-Server communication load* while it does not affect the *clients CPU load* and the *Client-to-Server communication load*.

• **FTP:** The second service we consider is FTP (File Transfer Protocol). FTP services are usually replicated by mediating between two opposite needs: from the server point of view, one would like to maximize FTP server throughput while, from the client point of view, one would like to minimize the mean download time for each client. To mediate between these opposite constraints, FTP servers are usually replicated through *mirroring* and then spread across the network in order to minimize the network distance between the replicated servers and the clients. Howewer this approach is not well supported from the client-side, the FTP protocol does not allow a client to retrieve from an existing FTP server a list of all its replicas. Some FTP client applications (as [Getright]) allow the user to manually input a list of replicated FTP servers and then choose, among them, the one with better round-trip-time (i.e. the time needed by an ICMP packet to go to destination an be acknowledged). Moreover, the Getright application periodically pings all the existing servers and automatically switches to the faster server even if there is a download in progress. This approach involves a fair amount of *Client-to-Server communication load* because each client needs to periodically ping all the existing servers.

• **Multiplayer online-gaming:** The online-gaming allows two or more players to attend a shared game session. Each player owns a copy of a same game and agrees with several other players to create a shared game session where all of them can attend.

To be able to cover a large amount of users, online-gaming providers usually replicate gaming servers in a hierarchical way.

As a consequence, the player who would like to find the better gaming server often needs to connect its client to each gaming server known searching for the server with the best performances. In the last years several solutions have been proposed for this problem, all these solutions work on the client side (as [Gamespy]) allowing a player to continuosly ping a list of known remote servers to find which of them is able to better serve a remote request. This approach is clearly not efficient since it contributes to increase the work load of servers. The requests sent by the client can themselves affect the server workload and compromise its ability to provide the service.

• **DBMS:** Parallelizing Database Management Systems is a known field of research whose aim is to improve performances of large DBMS by spreading load over several

computers. Oracle [Bamford98, Oracle8], IBM [Baru95] and others have realized parallel DBMS with different degrees of parallelism. Recent research [Exbrayat00] has proposed *coupled query evaluators* to add parallelism to an existing DBMS. Their system, Enkidu, is developed in Java and (while offering a efficient redistribution of queries to a pool of servers, thus allowing for a higher number of clients) has the drawback of a centralized *Server Module* that is in charge for spreading the queries to the cluster.

3. JSEB System

We assume that the services we want to scale can be replicated through several clusters of servers and that computers in the same cluster are connected by fast links while clusters are connected by slower links. JSEB system acts both on the:

• *Server side*: JSEB framework collects information about the state of all the replicated servers. This work is accomplished by running for each server an instance of the JSEB Server Monitor module. Each Server Monitor periodically retrieves information about the state of the server that is monitoring. These information are spread to all the other Server Monitors in the same cluster so that each Server Monitor knows the state of every other one. Moreover, for each cluster of Server Monitor we require the existence of a leader Server Monitor. The leader Server Monitor is in charge of providing information describing its own cluster state to the leader of other clusters (foreign clusters) and collecting cluster information reported by other leaders. A faulty leader triggers a simple leader election algorithm based on IDs. The information received from foreign clusters are spread from the leader to all the Server Monitors of its cluster. Finally, each Server Monitor manages a set of client connections, these connections are used to periodically spread the update state information to the client applications.

• *Client side*: JSEB framework allows a client application to know which is the state of all servers belonging to a set of clusters. Server state information are held by the Client Monitor modules. These modules are integrated into each client program, their purpose is to periodically receive update notification from their related Server Monitor. Using the Client Monitor module, the client is able at any time to identify the server machine whose state is optimal with respect to a given *service metric function.*

3.1. Service metric function

One of the main issues in scaling a service is rating the way a given server provides it. This rating can be accomplished by selecting a set of metrics that better describe the state of the server and combining them using a *service metric function*. Here is an overall classification of the standard metrics that can be defined in a client-server architecture:

• *Server application oriented metrics*: They measure the way the server application is performing including concepts like the number of active client connections, the number of the requests succesfully served in a unit of time and other service-specific parameters.

• *Host oriented metrics*: They measure how much loaded is a server machine including concepts like the amount of available resources, the CPU load[1] and the number of existing processes.

• *Connection oriented metrics*: They measure the quality of the communication links connecting the server machine to the clients including concepts like the bandwidth of the link, the round-trip time and the number of hops.

The *service metric function* weighs (ponders) the information provided by the selected metrics to estimate the performance of the given service.

An accurate *service metric function* must be built according parameters like the type of the service and the network topology, since these parameters may change it is not possible to define a standard one to be used in all cases. JSEB provides a set of pre-defined metric functions but the designer can specify particularly designed functions that take into account the semantic of the protocol.

3.2. JSEB architecture

We now describe JSEB architecture and functionalities. JSEB provides a set of general-purpose classes to be used in building scalable client-server services.

JSEB Server Monitor module.

The Server Monitor module is a Java object: we need to run an instance of Server Monitor for each server program we want to monitor. Server Monitor's functions are:

• *Collecting and spreading server state information*. Every Server Monitor instance holds a list of all the servers attached to the current cluster together with the related state information. Moreover, every Server Monitor holds also a list of all the servers belonging to foreign clusters together with their state information. Addition or deletion of a Server Monitor is notified to local cluster using a broadcasting algorithm, while it is notified to remote clusters by the leader. Information regarding the current state of the server are collected through the invocation of the method getLoad of the class Server Monitor, the coder chooses which metric information gather by providing an implementation of this method. State information are spread to all the Server Monitor.

• *Dealing with clients* . Server Monitor offers a service for accessing the state information collected by the JSEB framework. Client applications interested in obtaining servers' state information establish a connection with a

[1] Notice that, since we distinguish between application metrics and host metrics, we allow different services to run on the same machine.

reference Server Monitor. The client connection requests are managed by the Server Monitor using a rebalancing algorithm as shown in sec. 3.3. Moreover, Server Monitor periodically notifies to all its connected clients the state information list it collects.

JSEB Client Monitor module.

The Client Monitor module is a Java object. Clients to be attached to a Server Monitor cluster need to run an instance of Client Monitor in a separate Java thread. The Client Monitor functions are:

• *Receiving server state information*. Each Client Monitor needs to connect to a reference Server Monitor. By listening on this connection the Client Monitor receives from the Server Monitor the state information of all the servers belonging to all the available clusters. The Client Monitor holds an history of all the recent state information updates. The time between two update is variable, the Client Monitor can indicate to the Server Monitor a preferred update time rate, this indication is proposed by the client according to parameters like the type of service and the network bandwidth. If the reference Server Monitor fails the Client Monitor will try to connect itself to the least loaded Server Monitor known.

• *Selecting an optimal server*. At any time the client program can use its Client Monitor module to pick the address of the best server known. The server selection function is carried out by the method pickBest of the class Client Monitor. The selection function computes for each server a *service metric function* using the available state information, the name of this method is evalServer. After rating the state of each server, the Client Monitor module returns, as the best server, the one whose state is optimal. The default implementation of evalServer computes the identity function (i.e. the input is not processed) while the default implementation of pickBest computes the minimum function (i.e. the optimal server is the one with the minimum rating). These implementations can be overrided deriving the Client Monitor class and specializing it. Besides the address of the optimal server, the Client Monitor is also able to report the state of all servers and their rating.

3.3. JSEB Efficiency and Scalability.

The approach used by JSEB to solve the scalability issue requires a consistent communication overhead. This overhead could be critical especially when the number of Client Monitor or Server Monitor is very large, to face this problem JSEB implements several solutions:

• *adopting a hierarchical* Server Monitor *communication scheme.* While all the computers in a same cluster are connected via a fast communication link, computers in different clusters often can only use a slower communication link. In this scenario we use two distinct approaches for intra-cluster communication and for inter-cluster communication.

- *inter-cluster communication.* The leader of a JSEB cluster optimizes the communication with foreign leaders buffering all the relevant state information collected from the other Server Monitors in its cluster. Then, these information are sent to all the foreign leaders in one shot using a standard TCP socket connection.
- *intra-cluster communication.* Each Server Monitor in a cluster spread its state change information to the other Server Monitors using a UDP multicast socket connection. The UDP multicast communication allows a Server Monitor to notify its state change to all the other Server Monitors using a single transmission thus minimizing the amount of network load.

• *notifying only relevant state change.* Each time a Server Monitor runs the getLoad method to update its state information the new value is compared with the last known value. If the state change is relevant, than the new value is spread to the other Server Monitors and to the Client Monitor otherwise it is ignored. The default behaviour does not spread state changes that do not alter state information. This behaviour can be changed overriding the method compare of the class Server Monitor. In this way it is possible to define upon which conditions a state change has to be considered relevant.

• *adapting the update rate.* Initially, the Server Monitor tries to send state information to its Client Monitor with a fixed frequency. This frequency is adapted according to the current performance of the Server Monitor and to the speed of the communication links. In the first case the Server Monitor can slow down the update activity when there is too much load. In the second case the Client Monitor requests to the Server Monitor to change the update frequency. Notice that the programmer can choose the update frequency according to the nature of service provided.

Rebalancing algorithm and Fault-tolerance. As we have seen clients who use JSEB need to attach themselves to a Server Monitor using the Client Monitor module. On the server side, managing Client Monitor connections can be very expensive especially if the number of connected clients is very high. To better serve big amounts of client connections JSEB implements a rebalancing algorithm; each time a Server Monitor module receives a new client connection request it checks which is the minimum number of managed connections by any other Server Monitor. If the number of active connections is greater than the minimum number of a costant k, the connection request is sent to the least loaded Server Monitor. To make possible the rebalancing each Server Monitor needs to know the number of connections managed by all the other ones. This information is spread by the Server Monitors with the state information update.

This approach provides also a solution for handling Server Monitor failures. In these cases all Client Monitor that are connected to a faulty Server Monitor detect the failure and move themselves to the best Server Monitor they know. In this situation the rebalancing algorithm avoids a single Server Monitor to be overloaded by a large number of Client Monitor connections. Of course, it is possible that the best Server Monitor becomes crowded and, therefore, some of the Client Monitor will be moved toward the next best server and so on.

3.4. Client-side state information delivery.

According to the type of service a client could have need to update its notion of the servers state with a different time rate from the one used by the Client Monitor in its communication with the Server Monitor.

For example while using a pool of replicated Web servers, the state information could be effectively used each time a new HTTP request is made. Howewer, once run, a single request should not be affected by the servers state, this holds because the average size of the HTTP document is small so the cost to retrieve a document from a bad server is generally smaller than the one needed to redirect the request to a better server.

Instead, while downloading a file using replicated FTP servers we are interested in knowing at any time which is the fastest server available, if such server does not match the currently used server we can drop the existing connection and continue to download the file from the new server.

To face these cases JSEB offers two approaches, a *request* model and an *event-driven* model. In the *request* model each time the client program needs to request a service it uses its reference to the Client Monitor module to know which are the servers state. In the *event-driven* model, the client program register itself to a service offered by the Client Monitor module for the delivery of state information. Each time the Client Monitor will get an update from the Server Monitor, the client program will be notified by means of an JSEB update event.

The *request* model for state information delivery.

The Client Monitor module implements a set services for obtaining information about the state of the cluster of servers. The services offered by the Client Monitor are exported through the updateInterface. By requesting these services, a client program it is able to query the Client Monitor for information like the number of available servers, the IP address of the best server and the history of servers state information.

The *event-driven* model for state information delivery.

The Client Monitor module is able to spread the information received from a cluster of Server Monitor using a specialized version of Java events. Each Client Monitor holds a list of object references interested in receiving the information about the number and the state of the available servers. When the Client Monitor receives an update state information, it notifies this information to all interested clients objects by firing a JSEB update information event. The JSEB update information events are delivered to all interested clients using the standard Java event channel. The client objects that are interested in receiving JSEB events need to implement the jsebEventListener interface.

3.5. Application scenarios

• **File Transfer Protocol.** File Transfer Protocol is session oriented and therefore, the application of JSEB requires a certain amount of programming client-side that may involve integrating JSEB client into the existing FTP client.

In fact, limited coding is required if one wants just to add the capability to choose the "best" FTP server automatically (as in [Getright]). In this case, though, the FTP client is tied to the server for the whole session and, if the FTP server chosen becomes heavily loaded (or is not anymore the "best" server), then the (ordinary) client cannot automatically switch to another server.

If one wants to obtain a more efficient integration with JSEB (and a consequent wider scalability and load balancing) then a client must be developed (or reused) integrating mechanisms to preserve (and replicate) the state of the session after each command (such as current directory, authentication, etc.) so that, before each potentially "heavy duty" command, a *pickbest()* method would select the server designated to offer the best service. We notice that the policy to establish what command must be preceded by a *pickbest* selection can be either designed by the programmer and hardwired into the client (i.e. a policy as "*every GET/PUT command is heavy*"), chosen by the user (i.e. by configuring the client) or even adapted to the run-time performances of the replicated FTP server on the network of PCs (i.e. changing the treshold of file size which triggers the execution of *pickbest()*).

• **Multiplayer online-gaming.** The integration of JSEB into this architecture works both on the server-side and on the client-side. On the server-side, each replicated server runs a Server Monitor who gathers information about the workload of the service provider, spread this information to the other Server Monitor and notifies this information to the interest clients.

On the client-side, each players runs a Client Monitor to periodically receives update information about the state of each server. The significance of this approach is that it allows the servers to effectively control the flow of information, while in the client-side approach several clients could compromise servers activity by sending a large amount of requests, in our solution the servers push information to-

ward the clients, the clients are just listener.

• **Parallel DBMS.** As we said in the Introduction, it is possible to further increase scalability of Parallel DBMS as Enkidu [Exbrayat00] by adding JSEB into their system. This could be done at a very low cost since their system is entirely written in Java (with a JNI interface to some C code).

DBMS offers a good example of how JSEB generic service metric functions can be fruitfully istantiated for each service. For example, one may want to monitor servers not only according to their load (expressed by various parameters) but also as caching locality: a server can be particularly well-suited for an SQL query (no matter the load) since it already extracted a large part of the relevant data from the DB. By sending clients information on the last queries answered by each server, a Client Monitor can direct the query to the "best" server, that may not be the least loaded but that can quickly answer the query.

4. Conclusions

JSEB is a general framework to add scalability and fault-tolerance to existing systems whose servers are run on clusters of workstations. It is efficient and easily usable, moreover, it can deal with heterogeneous 3-level architectures by allowing (for example) to monitor and scale WWW and DBMS services at once. JSEB extensions to deal with multi-level distributed architectures (both server-side, as in the previous example, and client-side., by adding proxies) are straightforward and substantiate the claimed generality of JSEB.

Moreover, JSEB can be effectively used to build scalable services where server selection functions are very complex and specific to the used protocol. In these cases by migrating this computational load from the servers to the clients the *servers CPU load* is minimal. The *Server-to-Server communication load* and the *Client-to-Server communication load* required by the Client Monitor and Server Monitor activities can be fine tuned by properly setting the JSEB update rates. This option is useful when servers are distributed on a geographical network or when the *Servers-Clients ratio* is very low.

Further work is planned along several directions: on one hand, JSEB will be used to implement several scalable services such as FTP; on the other hand, we plan to include an adaptive mechanism to avoid communication overhead as the network performances degrade. Finally, interesting work is planned on the formalization of standard metrics to rate server performances for the most representative services.

References

[Amir96] Y. Amir, A. Peterson and D. Shaw. "*Seamlessly Selecting the Best Copy from Internet-Wide Replicated Web Servers* ". Proc. 12th International Symposium on DIstributed Computing, Andros Greece, September 24-26, 1998.

[Anderson96] E.Anderson, D.Patterson, E.Brewer. "The Magic Router: an application of fast packet interposing". Unpublished report, University of California, Berkeley, May 1996.

[Andresen96] D. Andresen, T. Yang, V. Holmedahl and O. Ibarra. "*SWEB: Towards a scalable WWW server on multicomputers* ". Proc. of the 10th International Parallel Processing Symposium (IPPS'96), Hawaii, April 1996.

[Baker99] S.M. Baker, B. Moon. "*Distributed cooperative Web servers*". Proc. of the 8th International World Wide Web Conference, pages 137-151, Toronto, Canada, May, 1999.

[Bamford98] R.Bamford, D.Butler, D.Klots et. al. "Architecture of Oracle Parallel Server". In Proc. of VLDB 98 (New York USA), pp.669-670, August 1998.

[Berners-Lee95] T. Berners-Lee. "*HyperText Transfer Protocol HTTP/1.0*". October 1995. HTTP Working Group Internet Draft

[Brisco95] T. Brisco. "*DNS support for Load Balancing*". April 1995. Network Working Group RFC 1794.

[Baru95] C.Baru, G.Fecteau, A.Goya et. al. "DB2 Parallel Edition". IBM System Journal, vol. 34 No. 2 PP. 292-322, 1995.

[Comer87] D.E. Comer. "*Internetworking with TCP/IP. Principles, Protocols and Architecture*". Prentice-Hall International, Inc., 1987.

[Crovella95] M.E. Crovella and R.L. Carter "*Dynamic Server Selection in the Internet*". Proc. of the Third IEEE Workshop on the Architecture and Implementation of High Performance Communication Subsystems (HPCS'95).

[Exbrayat00] M.Exbrayat, L.Brunie. "A PC-NOW Based Parallel Extension for a Sequential DBMS". Proc. of PC-NOW 2000, International Workshop on Personal Computer based networks Of Workstations, held with IPDPS 2000. Springer-Verlag LNCS series.

[Gamespy] http://www.gamespy.com

[Getright] http://www.getright.com.

[Holmedahl98] V. Holmedahl, B. Smith, and T. Yang. "*Cooperative Caching of Dynamic Content on a Distributed Web Server*". Proc of 7th IEEE International Symposium on High Performance Distributed Computing (HDPC-7) Chicago, IL USA July 28-31, 1998.

[Hunt98] G.H.Hunt, G.S.Goldszmidt, R.P. King, R. Mukherjee. "Network Dispatcher: a connection router for scalable Internet Services". Proc. of 7th International WWW Conference, Australia, 1998.

[Karger99] D. Karger, A. Sherman, A. Berkheimer, B. Bogstad, R. Dhanidina, K. Iwamoto, B. Kim, L. Matkins, Y. Yerushalmi. "*Web caching with consistent hashing*". Proc. of the 8th International World Wide Web Conference, pages 125-135, Toronto, Canada, May, 1999.

[Katz94] E.D. Katz, M. Butler, and R. McGrath. *"A Scalable HTTP Server: The NCSA Prototype*". Computer Networks and ISDN Systems 27 (1994) 155-164.

[Oracle8] Oracle. "Oracle Parallel Server: solution for mission critical computing". Tech. Rep. Oracle Corp. Redwood Shores, CA (USA) Feb. 1999.

[Sayal98] M. Sayal, Y. Breitbart, P. Scheuermann, R. Vingralek. "*Selection Algorithms for Replicated Web Servers*". Proc. of the Workshop on Internet Server Performance, 1998.

[Yoshikawa97] C. Yoshikawa, B. Chun, P. Eastham, A. Vahdat, T. Anderson, and D. Culler."*Using Smart Clients to Build Scalable Services*". Proc. of USENIX'97, 1997.

[Zhou93] Zhou, S., Wang, J., Zheng, X., and Delisle, P."*Utopia: A load Sharing Facility for Large, Heterogeneous Distributed Computer Systems.*" Software - Practice and Experience 23(12) December, 1305-1336 (1993).