ISSN 1392 – 124X INFORMACINĖS TECHNOLOGIJOS IR VALDYMAS, 2004, Nr. 3(32)

# ANALYSIS OF CORBA LOAD BALANCING STRATEGIES

**Arūnas Andriulaitis**

*Business Informatics department, Department of Applied mathematics, Kaunas University of Technology*
*Studentų st. 50-324, LT – 3031, Kaunas, Lithuania*

**Abstract**. Load balancing strategy describes how system handles and distributes workload among several machines. There are several load balancing strategies in CORBA technology. This article analyses architectures and workflows of the existing and theoretical load balancing strategies of CORBA. The article includes created mathematical aggregate model for load balancing strategies simulation, investigation results of strategies performance and results description and analysis. The mathematical aggregate model lets theoretically make conclusions of load balancing strategies fitness in various environments.

## 1. Indroduction

The count of online Internet services during the past decade has increased. For example, e-commerce systems and online stock trading systems concurrently serve many clients who transmit a large number of requests. When the number of clients and the number of connections to servers increases, servers become more loaded and this prolongs the response time of client operation. It is possible to invest in hardware, increase network speed or use load balancing as a problem solution. Just by using load balancing it is possible to get surprising results. By using load balancing, every server on the system receives similar load, therefore there are no overloaded servers.

There are many strategies of load balancing, several of them are listed below:

- IP, DNS based load balancing strategy [1]. This strategy is used in WWW systems.
- OS based load balancing strategy [1]. This strategy is mostly used in closed systems with large specific tasks.
- Middleware load balancing strategy [1,2,5]. Every technology has its own load balancing solution. IONA Orbix ORB and Inprise VisiBroker are the most popular realizations of CORBA ORB that have load balancing. These ORB realizations have similar load balancing strategies.

This article covers architecture and workflow analysis of all middleware CORBA load balancing strategies. The real test of such strategies requires many resources and is very expensive. In order to test fitness and effectiveness of the load balancing strategies, we created and described mathematical model for all load balancing strategies and provided analysis of statistical test results. Mathematical method provides statistical evidence of load balancing strategies effectiveness.

## 2. Existing CORBA strategies of load balancing

VisiBroker supports grouping function (clustering) [1, 5], which enables uniform objects to be grouped into one group and have one name. Every server stores its own objects into some group of objects. The group comprises many objects from different servers. All these groups are saved into Naming service reference table of one general ORB. In general, clients know only the object name, which is resolved by Naming service to object reference in one of servers, containing object in this object group. This way, Naming service can select one server from the group of servers, check server load and perform load balancing. When the client gets reference, it is redirected to another server (different than server with global Naming service), where the real object is deployed. This is the first and simpliest load balancing strategy.

## 3. Other CORBA strategies of load balancing

The above described VisiBroker load balancing strategy is able to check server load and perform load balancing only at the object name-reference resolution moment. This means that load balancing is provided on the client session basis, when client gets real object reference from CORBA Naming service. This strategy is called session based load balancing strategy.

It is possible to use another load balancing strategy, which provides load balancing on each object method usage not only at the moment of object creation [2]. This strategy is theoretical (does not have a realization). The client always communicates with servers through the load balancer (specific server). The load balancer contains only interfaces, it acts as proxy for real objects. Firstly, client gets reference to the object, which exists in the load balancer. When client calls this object method, the load balancer performs analysis of servers load and redirects client to server with the lowest load. This strategy is called per-method call based load balancing strategy.

Every strategy listed above has own disadvantages. Session-based load balancing strategy performs load balancing only at the moment of object creation and does not control servers load after that. Per-method call based load balancing strategy performs check of servers load everytime. This may be the time consuming operation. The client is always calling the method twice: once on load balancer and the second time on the server of the real object (second call is performed by ORB).

O.Othman, C.O'Ryan and D.C.Schmidt researched, described and suggested on-demand load balancing strategy [1]. By using this strategy it is possible to check server load at the moment of object creation and perform periodical load balancing and monitoring of servers load. When client wants to resolve object name-reference, it is calling load balancer Naming service methods. The load balancer analyses servers load and returns reference of server with the lowest load. This is the same functionality as in the session-based load balancing strategy. The load balancer periodically performs checking of servers load. If servers are loaded very differently, then the load balancer sends a command to the most loaded server to return the client to the load balancer. The client is redirected back to the load balancer. Now, the client ORB calls the same method of object, which is on the load balancer. The load balancer checks servers load and returns reference to the object, which is on server with the lowest load. The client is again redirected to the server, but this time the server has lowest load. This way, the load balancer always keeps servers loaded similarly and most effectively. These redundant operations (2 additional calls of method and two redirects) are performed by ORB and are transparent for client (client does not need to add any additional code).

Per-method and on-demand based strategies do not support object with the state. These strategies cannot be used with the objects, which have to keep their state.

The last two load balancing strategies do not have real performance check, when there are many servers and many clients. The real check of these systems is very expensive (requires many resources). Creating a mathematical model of systems and performing simulation could solve this issue.

## 4. A short introduction to aggregate systems

Aggregate is interpreted as an object, defined by a set of states *Z*, input signals *X*, and output signals *Y* and set of system events *E* [3, 4]. Aggregate functioning is considered in a set of time moments $t \in T$. The state $z \in Z$, inputs $x \in X$ and outputs $y \in Y$ are considered to be time functions. The system state consists of discrete $\upsilon(t)$ and continuous $z_v(t)$ components. The aggregate system has events $E = E' \cup E''$, where $E' = \{e_1', e_2', ..., e_n'\}$ is external events set and $E'' = \{e_1'', e_2'', ..., e_n''\}$ is the set of internal events. External events unambiguously correspond to the arrival of inputs $X \to E'$. Internal events unambiguously correspond to defined conditions on continuous components. For every event $e_i'' \in E''$ control sequences $\{\zeta_i\}$ are assigned, where $i = \overline{1,\infty}$, $\zeta_i$ have meaningful physical values of continuous or discrete components (i.e. this may mean operation duration time, speed, weight, etc). Transition operator *H* describes system states transitions after any event $e \in E$. Output operator *G* describes system outputs $y \in Y$ after any event $e \in E$.

General model of aggregate description is provided below:

$$\upsilon(t_m) \neq \varnothing,\ z_v(t_m) \neq \varnothing,$$
$$E' \neq \varnothing, E'' \neq \varnothing, \dot{z}_v = f(t, z_v(t_m)), t \in [t_m, t_{m+1}),$$
$$\upsilon(t) = const, when\, t \in [t_m, t_{m+1}),$$
$$z(t_{m+1}) = H(z(t_m), e_i),$$
$$y = G(z(t_m), e_i), e_i \in E' \cup E''$$

When the state of the system $z(t_m), m = 0,1,2,...$ is known, the moment $t_{m+1}$ of the following event is determined by a moment of input signal arrival to the aggregate or by the following equation:

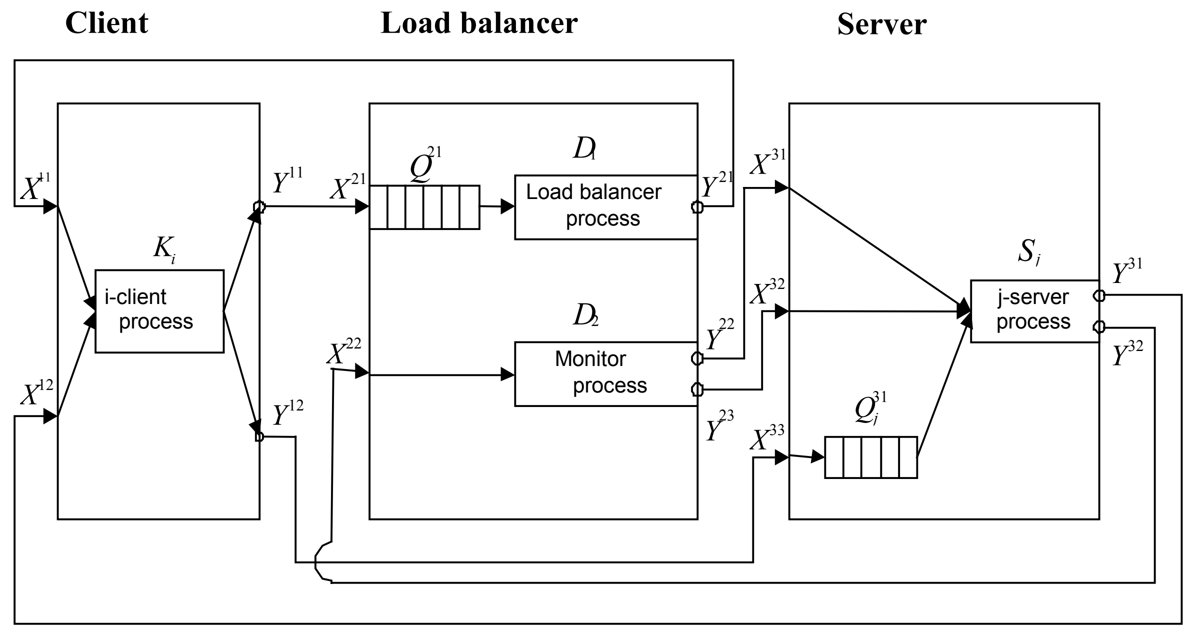
$t_{m+1} = \min\{t_{m+1}^{i*}\}, e_i \in E' \cup E''$, where $t_{m+1}^{i*}$ is time of every possible event $e_i$.

PLA (piece linear aggregate) is a simplified version of the general aggregate. In PLA $z_v(t)$ are linear continuous timers and the system coordinates $z_{v1}(t)$ and $z_{v2}(t)$ in the time intervals $[t_m, t_{m+1}]$, when *m*=0, 1, 2, ... vary according to the following equation:

$$\frac{dz_{vi}(t)}{dt} = -1,\ i = 1,2,...$$

## 5. Mathematical model of load balancing

We included only essential objects into the mathematical model: client aggregate, server aggregate, load balancer aggregate (load balancer and monitor) and channels. One client aggregate represents all real clients. It simulates N processes, where one process is one client. One server aggregate represents all real servers. It simulates M processes, where one process is one server. Further, we will write client not client aggregate. The same will be with server and load balancer.



The client process imitates the real client work-flow by sending signals through channel $Y^{12}$ to server and by waiting for the response. When the signal arrives, the server process imitates operation with some duration and returns answer trough channel $Y^{31}$. Before the act with the server, the client process sends a signal to load balancer (channel $Y^{11}$) to get identifier of server with the lowest load. After the client process receives the server identifier, it starts sending signals to the server. The client could be redirected to load balancer from the most loaded server (load balancer sends a command to the server to return client requests to load balancer). A formal description of the client aggregate is provided below:

1. Input.

Input signal (answer from the load balancer) to $X^{11} = (j,i)$, j – identifier of server process, i – identifier of client process.

Input signal (answer from the server) to $X^{12} = (i,k)$, k — type of signal (0 — continue, 1 — back to load balancer).

2. Output.

Output from $Y^{11} = (i)$ and $Y^{12} = (j,i)$.

3. Sets of events.

$E^{'} = \{e_1^{'}, e_2^{'}\}$, $e_1^{'}$ represents event, when signal to the input $X^{11}$ is got, $e_2^{'}$ represents event when signal to the input $X^{12}$ is got.

$E_i^{''} = \{e_{1i}^{''}, e_{2i}^{''}\}$, $i = 1..I$, $I$ is the total number of clients. $e_{1i}^{''}$ represents internal event, when the client is prepared to start a new session, $e_{2i}^{''}$ represents internal event, when the signal to the server is prepared.

4. Set of controling events.

$e_{1i}^{''} \Rightarrow \{\tau_{1i}^{r}\}_{r=1}^{\infty}$, $i = 1..I$, $\tau_{1i}^{r}$ is duration of any client idle time.

$e_{2i}^{''} \Rightarrow \{\tau_{2i}^{g}\}_{g=1}^{\infty}$, $i = 1..I$, $\tau_{2i}^{g}$ is duration of any client any operation.

5. States of aggregate.

$v_i(t_m) = \{sv_i(t_m),\ op_i(t_m),\ rlb_i(t_m),\ rsv_i(t_m)\}$, $i$ = 1..$I$, $sv_{i,}(t_m)$ identifier of server process, $op_i(t_m)$ is the count of operations in the client session. The client session consists of a number of request operations, which are performed without stopping. If $rlb_i(t_m)$=1, then the client needs to get identifier of server from the load balancer. If $rlb_i(t_m)$=0, then the client continues sending requests to the server. If $rsv_i(t_m)$=1, then the client needs to start sending requests to the server. If $rsv_i(t_m) = 0$, then the client needs to define a new count of operations in session $op_i(t_m)$.

$z_i(t_m) = \{W_i(e_{1i}^{''}, t_m), W_i(e_{2i}^{''}, t_m)\}$. $i$ = 1..$I$ $W_i(e_{1i}^{''}, t_m)$ represents the end of client idle operation (between sessions), $W_i(e''_{2i}, t_m)$ represents the end of client ordinary operation (request to the server).

6. Starting state.

$$v_i(t_0) = \{-1, -1, 0, 0\},\ i = 1..I$$

$$z_i(t_0) = \{\infty,\ \tau^1_{2i}\},\ i = 1..I$$

7. Some operators of transitions.

$H(e'_1)$ represents the transition of system state, when external event is got $e'_1 \rightarrow X^{11} = (j, i)$ from the load balancer.

New identifier of the server is stored.

$$sv_i(t_{m+1}) = j$$

New count of operations is calculated.

$$op_i(t_{m+1}) = \begin{cases} RANDOM,\ if\ rlb_i(t_m) = 0 \\ op_i(t_m),\ if\ rlb_i(t_m) = 1 \end{cases}$$

Flag is set to continue session.

$$rlb_i(t_{m+1}) = 0$$

Flag is set to generate or not the count of operations.

$$rsv_i(t_{m+1}) = \begin{cases} 1, if\ rlb_i(t_m) = 1 \\ 0, if\ rlb_i(t_m) = 0 \end{cases}$$

The time moment when the client request will be prepared is set.

$$W_i(e''_{2i}, t_{m+1}) = t_m + \tau^g_{2i}$$

The time moment when the client will start new session is unknown.

$$W_i(e''_{1i}, t_{m+1}) = \infty$$

$H(e''_{2i})$ represents the transition of system state, when internal event $e''_{2i}$ occurs.

The same identifier of the server is stored.

$$sv_i(t_{m+1}) = sv_i(t_m)$$

The count of operations is decreased (–1 means session is stoped).

$$op_i(t_{m+1}) = \begin{cases} -1, if\ rsv_i(t_m) = 0\ and\ rlb_i(t_m) = 0 \\ op_i(t_m) - 1,\ if\ rsv_i(t_m) \neq 0\ and\ rlb_i(t_m) \neq 0 \end{cases}$$

Flag is set to continue sending requests to the server.

$$rlb_i(t_{m+1}) = rlb_i(t_m)$$

Flag is set to generate or not new count of operations.

$$rsv_i(t_{m+1}) = \begin{cases} 0,\ rlb_i(t_m) = 0 \\ 1, rlb_i(t_m) = 1 \end{cases}$$

The time moments, when any internal event will occur are unknown.

$$W_i(e''_{2i}, t_{m+1}) = \infty$$

$$W_i(e''_{1i}, t_{m+1}) = \infty$$

The client sends new signal *(j, i)* to the server if flag $rlb_i(t_m) = 0$ is set and the client sends new signal *(i)* to the load balancer if $rlb_i(t_m) = 1$ is set.

$$Y^{12} = (j, i),\ if\ rlb_i(t_m) = 0$$

$$Y^{11} = (i),\ if\ rlb_i(t_m) = 1$$

When load balancer receives client request for identifier of server, it returns identifier of server with the lowest load (channel $Y^{21}$). This aggregate has an additional process, which periodically monitors servers load (channels $Y^{22}$, $Y^{23}$) and sends commands to servers to return client to load balancer. In this case, the client at any server call could be redirected back to load balancer just to retrieve a new identifier of server.

The above described aggregates imitate all three strategies of load balancing. It is enough for session-based load balancing strategy imitation that client calls load balancer once. The client has to perform only one operation per session and immediately start again a new session for per-method call based load balancing strategy imitation. In order to imitate the third load balancing strategy this model does not need any change.

In order to detune workload of the system, the client must use a different number of calls per session or/and operation duration must be different. This model imitates only a different number of calls per session.

## 6. Results of experimental simulation

The experimental environment was the following: 100 clients, 10 servers, one part of the client processes perform 10 operations per session and the second part perform 1 operation. Average duration of the client operations and servers load are illustrated in Figures 1 and 2.

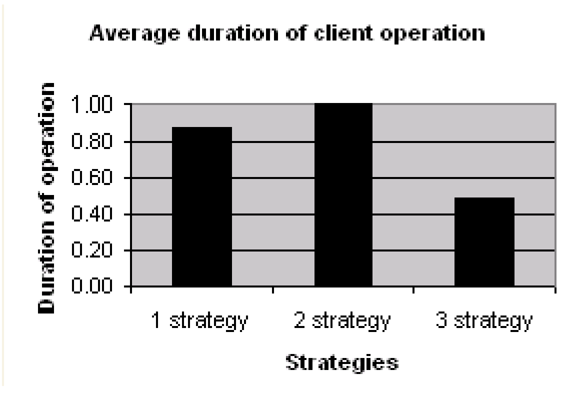


**Figure 1**

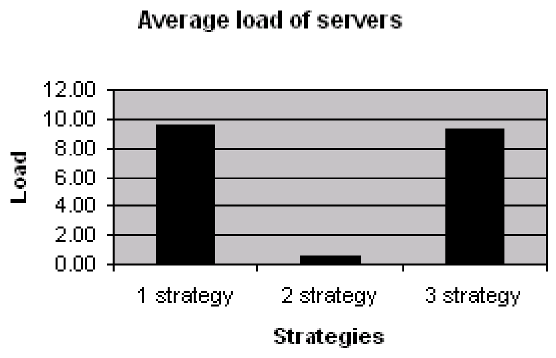


Figure 2

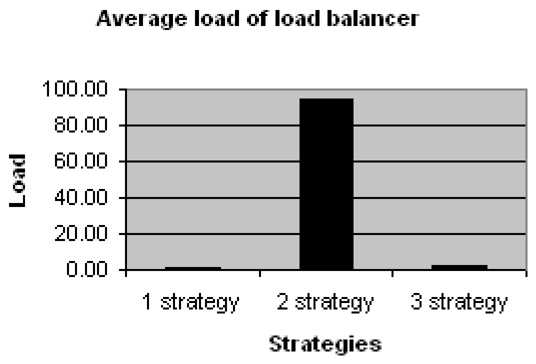


Figure 3

The servers and load balancer load is represented as length of queue of waiting requests; duration of client operation is relative and has no physical meaning, but it is used to compare effectiveness of different strategies.

- Session-based load balancing strategy is quite effective. Every server has about 10 client requests on the average at any time moment.
- Per-method based load balancing strategy is very ineffective. Architecture of this strategy overloads load balancer a lot. Operation of load balancer is time expensive and prolongs every client operation. The load balancer is overloaded, but servers are not.
- On-demand based load balancing strategy is the most effective. The client average operation is shortest. Architecture of this strategy allows to distribute server load effectively.

## 7. Conclusion

The most effective strategy is the on-demand based load balancing strategy. But this strategy could be used only with the objects, which do not have states. If server objects need to have their state, then the session-based load balancing strategy is more appropriate.

## References

**[1]** **O. Othman, C. O'Ryan, D. C. Schmidt.** The Design and Performance of an Adaptive CORBA Load Balancing Service. 2001, *http://www.cs.wustl.edu/~schmidt/PDF/load_balancing.pdf*

**[2]** Inprise. *VisiBroker documentation*. *http://info.borland.com/techpubs/books/vbj/vbj45/framesetindex.html*

**[3]** **K. Wang, H.Pranevicius.** Applications of AI to Production Engineering. *Kaunas University of Technology Press*, 2000, 274-320.

**[4]** **H. Pranevičius.** Модели и методы исследования вычислителных систем. 1982, *Mokslas*.

**[5]** **M. Lookwave.** The Three Tier architecture. *Expert-Soft Corporation*, 1996.