Towards an Efficient Cluster-based E-Commerce Server

Victoria Ungureanu
Rutgers University
180 University Ave., Newark, NJ 07102,USA
email: ungurean@cs.rutgers.edu

Benjamin Melamed
Rutgers University,
94 Rockafeller Rd., Piscataway, NJ 08854, USA
email: melamed@rbs.rutgers.edu

Michael Katehakis
Rutgers University,
180 University Ave., Newark, NJ 07102, USA
email: mnk@andromeda.rutgers.edu

Abstract

Cluster-based server architectures combine good performance and low cost, and are commonly used for applications that generate heavy loads. Essentially, a cluster-based server consists of a front-end dispatcher and several back-end servers. The dispatcher receives incoming requests, and then assigns them to back-end servers, which are responsible for request processing. The many benefits of cluster-based servers make them a good choice for e-commerce applications as well. However, applying this type of architecture to e-commerce applications is hindered by the fact that e-commerce clusters have the additional task of verifying that requests comply with contract terms. The problem is further complicated by the fact that contract terms may be expressed as functions of dynamic, mutable states.

The problem addressed in this paper is the effective assignment of e-commerce requests, such that the load is balanced among back-end servers and request validation is efficient. To this end, we propose a policy called `TDA` *(Type Dependent Assignment), which takes account of the type of contracts. Under* `TDA`*, stateless contracts are replicated on all back-end servers. In contrast, a stateful contract,* `C`*, is preassigned to a designated back-end server, called the base of* `C`*, which is responsible for maintaining the state of* `C`*. The operation of* `TDA` *may be broadly outlined as follows. A request governed by a stateless contract is assigned to the least loaded server. In contrast, a request governed by a stateful contract is assigned to its base, if the base is not overloaded, and to the least loaded server, otherwise.*

1. Introduction

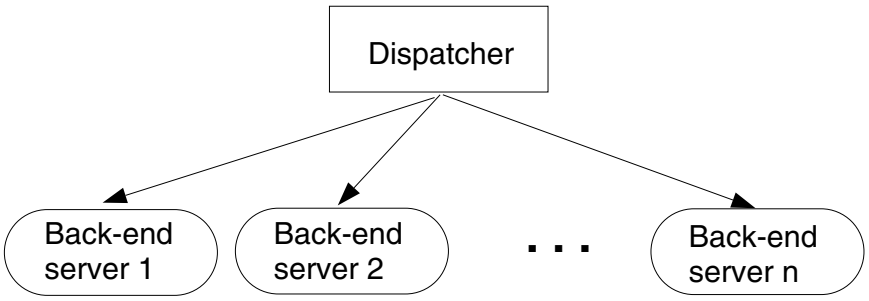
Motivated by the need to cut costs and increase competitiveness, more and more enterprises are migrating to on-line transactions with trading partners [5]. Among the problems inherent in this migration, none is more serious than the difficulty of controlling the activities of the disparate agents involved in e-commerce.

Trading relations between enterprises are based on mutually agreed contracts. Generally, contracts enumerate agents authorized to participate in transactions, and spell out such things as the rights and obligations of each partner and the terms and conditions of trades. Moreover, at any given time an enterprise may be bound simultaneously by several contracts. For example, Ford has approximately thirty thousand suppliers, each operating under a different contract, and General Motors has about forty thousand [6] (both companies have announced their intention to perform their inter-enterprise business on-line).

As an example of a contract, consider that agents in a client enterprise (say Ford) may purchase merchandise (tires, in this example) from a supplier enterprise (say Firestone), under the following terms:

- *Firestone honors purchase orders (POs) issued by Ford, for up to a cumulative value, called* blanket*, of 10,000 tires each month at a prescribed price of $25/tire.*
- *Only agents duly certified as purchase officers by* `ford_CA` *(a designated certification authority of the client enterprise) may issue POs; only agents duly certified as sale representatives by* `firestone_CA` *(a designated certifying authority of the supplier enterprise) are authorized to respond to POs.*

Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER'03)
0-7695-2066-9/03 $17.00 © 2003 **IEEE**



Contract terms may take into account dynamic information, referred to as *state*. The state, which *evolves* over the lifetime of the contract, might consist of various contract phases, the state of transactions regulated by the contract, or past actions of various agents. A contract whose terms are expressed in terms of state is called *stateful*; a contract that consists only of immutable terms is called *stateless*. For example, the contract presented above is stateful, and its state is the value of the blanket. The validity of a PO depends on the current value of the blanket: a PO is valid provided the number of tires requested does not exceed the current value of the blanket. Furthermore, the blanket value is decreased whenever a valid PO is issued.

The satisfactory execution of e-commerce applications depends on the performance of the corresponding servers. To date, applications that generate heavy loads, commonly use *cluster-based* server architectures [3, 9, 11] that combine good performance and low cost. Essentially, a cluster-based server consists of a front-end dispatcher and several back-end servers, as depicted in Figure 1. The dispatcher receives incoming requests, and then assigns them to back-end servers, which are responsible for request processing. This type of architecture achieves transparency and a certain degree of scalability and fault-tolerance by decoupling request receiving from processing.

The many benefits of cluster-based servers make them a good choice for e-commerce applications. However, applying this type of architecture to e-commerce is hindered by the fact that an e-commerce cluster needs to verify that requests comply with the governing contracts. To emphasize the inherent difficulties of this problem, we consider several straightforward solutions and discuss their pitfalls.

First, consider replicating all contracts on all back-end servers. This solution has the potential advantage of load balancing, because a request can be assigned to the least loaded server with the assurance that that server can verify compliance. However, it may lead to poor performance, because contract states have to be maintained consistent. Unfortunately, achieving consistency through traditional algorithms, such as two-phase commit, would strain back-end servers: *every change* of the state of *every contract* would have to be committed by *all back-end servers*, even though they might never serve a request regulated by some of the contracts.

Secondly, assume that requests governed by a given contract are assigned only to a pre-specified back-end server. Because a contract is served by only one back-end server, this solution does not require maintaining the consistency of contract states. However, it might lead to load imbalance. Finally, we mention that delegating request validation to the dispatcher (or some other central entity, for that matter) defeats the purpose of using a distributed architecture in the first place.

In order to implement cluster-based architectures in e-commerce setting, we propose a policy that combine good load balancing with efficient validation of requests. The proposed policy, called `TDA` (Type Dependent Assignment), takes account of contract types. Under `TDA` the contract rules (immutable part) are replicated on all back-end servers. In contrast, the state of any stateful contract `C`, is preassigned to only one designated back-end server, called the *base* of `C`.

The broad outline of `TDA` can be described as follows. For every incoming request, the dispatcher determines the type of contract governing it (stateless or stateful). If the contract is stateless, the dispatcher assigns the request to the least loaded server (determined according to a prescribed load metric). However, if the contract is stateful, the dispatcher assigns the request to the base of `C`, whenever the base is not overloaded. Note, that since the base maintains the state of `C`, no extra communications are needed in this case. In contrast, if the base is heavily loaded, the dispatcher reassigns the base to the least loaded server. We point out that only in this case, the state is sent to the new base and communication overhead is incurred. To support `TDA` implementation, we assume that contract formulation explicitly states contract type, and distinguishes between mutable and immutable parts of a contract.

The rest of the paper is organized as follows. Section 2 describes the contract enforcement mechanism. Section 3 describes in detail the `TDA` policy, and Section 4 concludes the paper.

2. Contract Enforcement

There has been a growing interest in supporting e-commerce contracts, and a variety of different, and quite powerful, enforcement mechanisms have been devised (see, for example, [1, 2, 4, 7, 8, 12]. However, to the best of our knowledge, none of the frameworks proposed so far considered applying cluster-based architectures to e-commerce applications, nor to any applications governed by stateful policies.

In order to apply cluster-based architectures to e-commerce applications we assume enforcement is carried out by generic contract-engines. Contract engines are generic tools that can verify certificates, and interpret and carry out the set of rules formalizing contract terms. In the assumed enforcement mechanism each back-end server has an associated contract-engine which maintains the immutable part (type and rules) of all contracts, regardless of their type. Moreover, a contract-engine maintains the state of those contracts for which it serves as base (the assignment of contracts to their base will be described in the next section).

Given these assumptions enforcement is carried out as follows. Consider that the dispatcher receives a request R, and assigns it to a back-end server S, having an associated contract-engine CE. Then the following steps will be taken:

1. CE determines the contract C governing request R. For example, the URL may serve as reference to the contract id.
2. If the request is accompanied by certificates then, CE verifies each certificate presented by the requester.
3. CE checks whether R is authorized by C. If this is the case, R will be processed by back-end server S; otherwise, the request is discarded.

We point out that to insure that the state of a contract C is maintained consistent, requests regulated by C should be handled *sequentially* in chronological order of their occurrence.

3. The TDA Policy

The TDA policy makes the following assumptions and stipulations. First, TDA stipulates that the initial choice of bases by the dispatcher is made in Round-Robin manner. Second, TDA assumes that at any given time, the dispatcher knows the number of requests pending completion at each back-end server. These values are estimated by the number of active server connections and are used as a measure of server loads. This is a practical assumption, because the dispatcher is responsible for handling connections and passing incoming data from clients to back-end servers. Consequently, the dispatcher must keep track of open and closed connections at each server (cf., for e.g., [11, 13]). Finally, TDA assumes that the dispatcher has a parameter, AC, defined as the maximum number of requests (expressed as the number of active connections) that a server may process concurrently (to avoid excessive delays).

Subject to the assumptions and stipulations above, TDA operates on a request R governed by a contract C as follows:

1. *When* R *arrives at the dispatcher, the dispatcher determines the type of* C.
2. *If* C *is stateless, the dispatcher assigns* R *to the least loaded the server, where the load of a server is given by its number of active connections.*
3. *If* C *is stateful, the dispatcher determines the base* S *of* C. *If the number of active connections of* S *is smaller than* AC, *then the request is assigned* S.
4. *However, if the number of active connections of* S *exceeds* AC, *the dispatcher determines the least loaded back-end server, say* S′. *If* S′ *is less loaded than* S, *then the dispatcher marks* S′ *as the base of* C *and assigns the request to it. The dispatcher further notifies* S *of the change. After* S *finishes processing all requests in its queue that are governed by* C, *it sends the state of* C *to* S′.

We point out that base reassignment is motivated by the fact that the initial assignment is made in Round-Robin manner, which is only a crude means for load balancing. Even though all back-end servers act as bases for the same number of contracts, some of them may get an unusually high number of "active" contracts, which govern a large fraction of the requests received in a certain time interval, and consequently their performance may degrade.

Changing bases incurs communication overhead, because the new assignment needs to be communicated to the old and new bases, and the state needs to be transferred to the new base. To balance the benefit of base change with its cost, the dispatcher does not change the base every time there is a server which is less loaded than the base. Rather, a base reassignment is carried out only when: (a) the number of active connections of the current base exceeds the prescribed threshold, AC, and (b) there is a server which is substantially less loaded than the current base.

Discussion We mention that in devising TDA we have been heavily influenced by the LARD (Locality-Aware Request Distribution) policy [10]. The goal of LARD is to assign HTTP requests so that access to disk — an expensive operation — is kept low. To this end, the dispatcher maintains for each document a set of back-end servers that have the document cached with high probability, and assigns a request for a document to the least loaded server from this

set. There are, however, several important differences between LARD and TDA, and these stem from the type of the underlying applications:

- LARD treats all requests similarly in the sense that it assigns a request for a document to the least loaded base in the set. In contrast, TDA treats requests governed by stateless and stateful contracts differently.
- Under LARD a document can have several bases, whereas under TDA a stateful contract has to have a unique base; moreover, under LARD each document has a base, whereas under TDA only stateful contracts need to have a base.

We conclude this discussion by pointing out that TDA is more amenable to practical implementation than LARD. This is so, because a Web cluster may serve a very large number of distinct documents, and consequently the table storing the mapping between documents and bases can grow quite large. It follows that maintaining and consulting this table can be expensive operations. In contrast, contracts are substantially less numerous than documents, and thus, the corresponding mapping table is much smaller. Therefor, we expect that mapping contracts to bases is not onerous.

4. Conclusion

In this paper we propose an assignment policy for e-commerce clusters, called TDA, which takes into consideration request validation. However, this policy is probably not efficacious for every workload pattern. For example, if the workload is light, there is a high probability that whenever a request arrives there is an idle server on which it can be served without delay. In this case, traditional policies, like LC or Round-Robin, may perform comparably to TDA or even better. A real challenge is to find a way to continually adapt the dispatcher policy to changing workload patterns.

References

[1] S. Abiteboul, V. Vianu, B. Forham, and Y. Yesha. Relational transducers for electronic commerce. In *Symposium on Principles of Database Systems*, pages 179–187, June 1998.

[2] A. Abrahams, , and J. Bacon. Representing and enforcing e-commerce contracts using occurrences. In *Proc. of the 4th International Conference on Electronic Commerce Research (ICECR4)*, pages 59 – 82, Dallas, Texas, USA, November 2001.

[3] V. Cardellini, M. Colajanni, and P. S. Yu. Dynamic load balancing on web-server systems. *IEEE Internet Computing*, 3(3):28–39, 1999.

[4] A. Dan, D. Dias, R. Kearny, T. Lau, T. N. Nguyen, F. N. Parr, M. W. Sachs, and H. H. Shaickh. Business-to-business integration with tpaML asnd a business-to-business protocol framework. *IBM Systems Journal*, 40(1):68–90, 2001.

[5] Economist. E-commerce (a survey). pages 6–54. (The February 26th 2000 issue).

[6] Economist. Riding the storm. pages 63–64. (November 6th 1999 issue).

[7] B. N. Grosof, Y. Labrou, and H. Y. Chan. A declarative approach to business rules in contracts: Courteous logic programs in XML. In *Proceedings of the first ACM Conference on Electronic Commerce (EC99)*, November 1999.

[8] S. Ketchpel and H. Garcia-Molina. Making trust explicit in distributed commerce transactions. In *Proceedings of the International Conference on Distributed Computing Systems*, pages 270–281, 1996.

[9] R. Lavi and A. Barak. The home model and competitive algorithms for load balancing in a computing cluster. In *The 21st Intl. Conf. on Distributed Computing Systems (ICDCS'01)*, Apr. 2001.

[10] V. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, and E. Nahum. Locality-aware request distribution in cluster-based network servers. In *Proceedings of the Eighth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VIII)*, 1998.

[11] V. Pai, P. Druschel, and W. Zwaenepoel. Flash: An efficient and portable web server. In *Proceedings of the USENIX 1999 Annual Technical Conference*, 1999.

[12] M. Roscheisen and T. Winograd. A communication agreement framework for access/action control. In *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, California, May 1996.

[13] T. Y.M. and R. Ayani. Comparison of load balancing strategies on cluster-based web servers. *Simulation, The Journal of the Society for Modeling and Simulation International*, 77(5-6):185–195, November-December 2001.