Fast Information Retrieval in the Open Grid Service Architecture

Tobias Berka and Marian Vajteršic

Department of Computer Sciences
University of Salzburg
Austria
tobias.berka@gmx.net, marian@cosy.sbg.ac.at

Abstract. Information retrieval offers resource discovery mechanisms for unstructured information and has thus been identified as a standardization goal by the open grid forum. We argue that an integration of information retrieval into the infrastructure is not only an interesting prospect for grid users, but is in fact necessary because the batch processing approach supported by the open grid service architecture is at odds with the requirements of online query processing. The cost of staging the search indices to an allocated compute node to answer sporadic but frequent search queries is prohibitive. We advocate the use of web services as a cross site messaging mechanism and discuss the alternatives. To investigate, we have designed and built a prototype system for grid image retrieval. Unfortunately, the statelessness and isolation of web services proved problematic for our purposes, but we present a software architecture that can efficiently overcome these issues.

Keywords: information retrieval, Grid computing, distributed computing, parallel algorithms, open Grid service architecture, web services

1 Introduction

If multiple organizations decide to join forces and create a virtual organization (VO) to pool and share their resources, it is clear that we require means to discover resources of interest, including large collections of images or texts. Two key issues complicate the situation: the documents are inherently distributed and incoming queries must be answered sporadically and frequently. In research, expensive tasks of conventional information retrieval systems have successfully been deployed as batch jobs on the grid [1] or in more intricate architectural forms using workflow engines [2], but the biggest challenge is to accelerate the query processing. For conducting information retrieval as a batch job, it is necessary to move the entire index back and forth between the storage nodes and the compute nodes. To eliminate the problem of index migration, we argue that means for information retrieval should be integrated into the grid infrastructure as a distributed, cross-site activity.

To comply with the overall direction taken by the Open Grid Forum (OGF), we should design a service-oriented architecture using web services as a means of communication between nodes. Another approach would be to use methods for the cross-site deployment of grid-aware implementations of the message passing interface (MPI) [4], which provide better communication performance and allow use of the popular MPI interface. But this limits the openness of the distributed retrieval system because all local implementations are forced to use a specialized MPI implementation. Others have investigated the use of middleware for service-oriented architecture other than web services for information retrieval systems, e.g. the OSIRIS middleware framework [5], but these are often available only in research implementations and do not enjoy widespread use. We believe that we should choose the first option, comply with the OGSA and use web services as a means of communication despite the increase in cost of cross-site messaging.

2 Fast Image Retrieval for e-Science Grids

One plausible scenario for grid information retrieval is the retrieval of images in a high-performance grid for e-science applications. We need a very high degree of retrieval accuracy and a complete coverage of the available documents, because the users of such systems require reliable search results for their work. In addition, we seek to obtain a maximum of performance in order to provide a very responsive search engine – key factor in providing a satisfying user experience [6].

For the sake of efficiency, our system is designed for distributed, parallel retrieval with distributed control. The retrieval activity is implicitly controlled by the exchange of messages and we consequently do not require a coordinating host. To obtain simplicity in the design and efficiency in the implementation, we decided to choose a specific retrieval model: the vector space model. Members of the VO can all submit documents to the distributed system, but they do so through a single, designated server. This design decision greatly simplifies the connection from the clients to the distributed system. Since the actual workload is carried out primarily by the back-end hosts, a well-designed front-end can easily handle large numbers of requests.

The principle distribution scheme is a document partitioning – the documents are distributed amongst the hosts of the system. In order to conduct a complete, exhaustive query over the entire document collection, we proceed in three steps: we distribute the query amongst all hosts, score and sort all local documents and merge-sort the local results to form a global hit list. Both the query distribution and merge-sort activities use a communication pattern of a flat binary tree, allowing us to reach all hosts in a logarithmic number of serial messaging steps, which is an important feature for a distributed cross-site activity. The message content for query merging consists only of document-similarity-pairs. If the documents are evenly distributed, the total serial message size is asymptotically linear in the number of documents – a clear advantage over the traditional approach, where term-by-document matrices have to be transmitted.

During the implementation of our prototype system, we had to overcome one major obstacle: the statelessness of web services and the isolation of the web service containers.

3 Overcoming Statelessness and Isolation of Web Services

In theory, web services are designed to be closed operations without any protocol-specific state, which are executed within the isolation of a web-service container. But for many applications, web services must operate on the application's state and these principles are being subverted. The most common way is to store the application state in a relational database and use a database connectivity driver for manipulation. As a more structured approach, the web service resource framework (WSRF) is a collection of XML-based standards for the creation, usage and management of state information for web services using persistent storage. Similarly, a web service implementation could simply use the file system to store its state in a custom file format. For a distributed, parallel information retrieval system, statelessness and isolation are highly problematic. The key reason for realizing information retrieval as a service was to prevent index migration for efficiency. Now, web services create a similar problem: we must avoid moving the index to and from expensive persistent storage. Therefore, we decided to use remote procedure calls (ONC-RPC). The web service simply reformats the data to data structures suitable for RPC transmission, dispatches a call to the RPC handler, which executes the implementing function for the call.

The implementation of each remote procedure places the message content in one of two message queues: one for processing requests and another for delivery of intermediate results. These two queues are shared between a messaging thread for the execution of the remote procedure calls and an application thread, which is responsible for the primary retrieval functionality. The request queue is used to keep track of all distributed operations over the shared document collection, such as managing documents and initiating queries. The application thread executes a loop, which blocks until a processing request is received, enters the function implementing the corresponding functionality and repeats this loop until a shutdown-request arrives. Within any such activity, which corresponds to a single function in the implementation, the delivery queue is used to receive incoming data. It is operated in a push-mode: the sender sends without waiting for a request, and the receiver is simply blocked when it attempts to pop a message from the empty queue. In the distributed merge-sort activity every second host begins simply by sending its local result to the delivery queue of every first host. This host then pops these results from the queue and merges it with its own results, which are then sent further along the flat tree topology. Differences in processing time are automatically balanced where possible, due to the asynchronous, push-based semantics of the queue. Similarly, processing requests are automatically received and buffered by the whole ensemble of hosts, allowing them to act as a persistent parallel service to the outside world. Figure 1 depicts the basic operation of these queues.

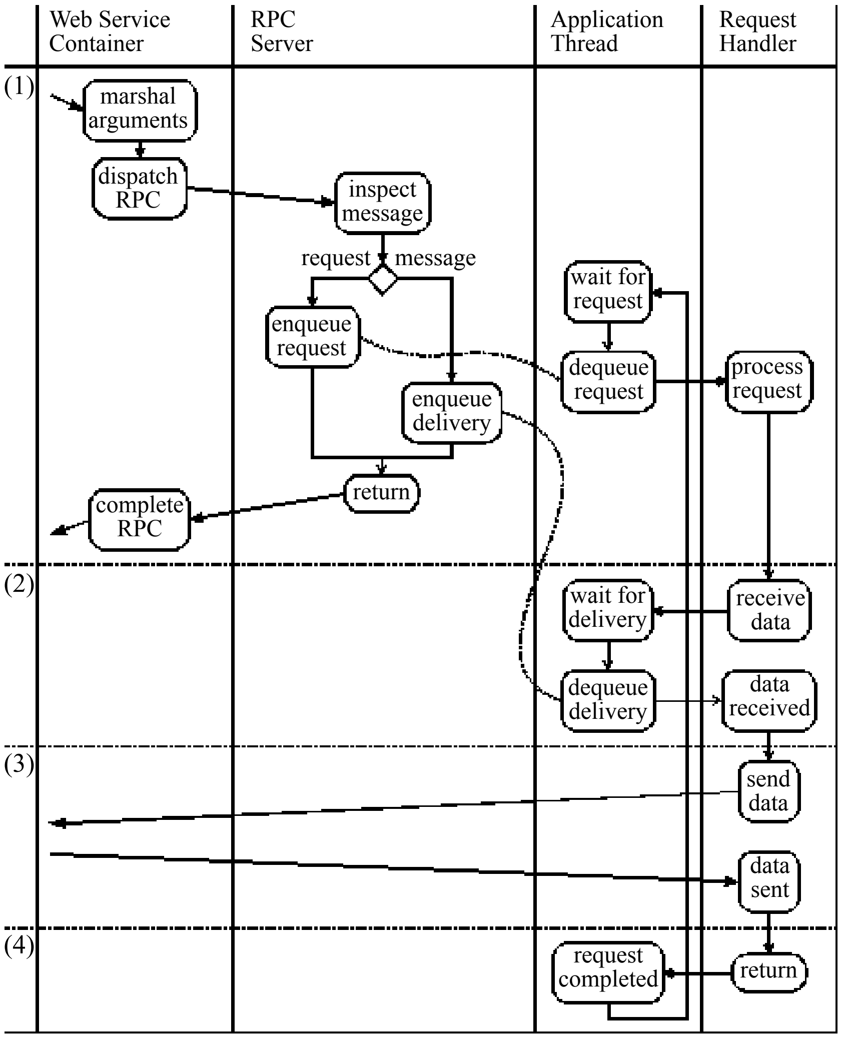

Fig. 1. Swim lane flowchart illustrating the use of the request and delivery queue to overcome statelessness and isolation of web services. This diagram depicts an incoming request and associated handler invocation (1), asynchronous receive operation for incoming data (2) and silent handler termination (4).

4 Conclusions

We have argued that information retrieval for grids is best realized as a persistent parallel service and as part of the grid infrastructure to prevent performance degradation due to costly migration of search indices. To comply with the open grid service architecture, we are using web services as a primary means of communication. Based on our prototype implementation, we have designed and implemented a software architecture that allows us to escape the statelessness and isolation of the web service container by using remote procedure calls to an RPC server – on the same computer or within the same local area network.

We believe that our fundamental approach to grid integration for our fast retrieval system can also be useful in other situations, where sporadic but frequent queries must be answered within a minimal response time. The use of remote procedure calls and shared in-memory message queues lends itself well to other applications that benefit from keeping the data model in memory. In future work, we will extend our current architecture to allow for multiple, parallel application threads to mitigate the high communication costs across high-latency network links through multi-programming.

References

1. Hughes, B., Venugopal, S., Buyya, R.: Grid-based Indexing of a Newswire Corpus. In: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing, pp. 320—327. Washington, IEEE Computer Society (2004)
1. Larson, R.R., Sanderson, R.: Grid-based Digital Libraries: Cheshire3 and Distributed Retrieval. In: Proceedings of the 5th ACM/IEEE-CS Joint Conference on Digital Libraries, pp. 112—113. New York, ACM (2005)
2. Foster, I., Kishimoto, H., Savva, A., Berry, D., Djaoui, A., Grimshaw, A., Horn, B., Maciel, F., Siebenlist, F., Subramaniam, R., Treadwell, J., von Reich, J.: The Open Grid Services Architecture, Version 1.5. Online publication of the OGF, available at http://www.ogf.org/documents/GFD.80.pdf (retrieved June 2010)
3. Coti, C., Herault, T., Peyronnet, S., Rezmerita, A., Cappello, F.: Grid Services for MPI. In: Proceedings of the Eighth IEEE International Symposium on Cluster Computing and the Grid, pp. 417—424. Washington, IEEE Computer Society (2008)
1. Brettlecker, G., Milano, D., Ranaldi, P., Schuldt, H.: DelosDLMS - A Next-Generation Digital Library Management System. In: Proceedings of the 14th International Conference of Image Analysis and Processing - Workshops, pp. 83—88. Washington, IEEE Computer Society (2007)
2. Chowdhury, A., Pass, G.: Operational Requirements for Scalable Search Systems. In: Proceedings of the Twelfth International Conference on Information and Knowledge Management, pp. 435—442. New York, ACM (2003)