Speak<geek>
Tech Brief

3

RichRelevance Distributed Computing:

creating a scalable, reliable infrastructure

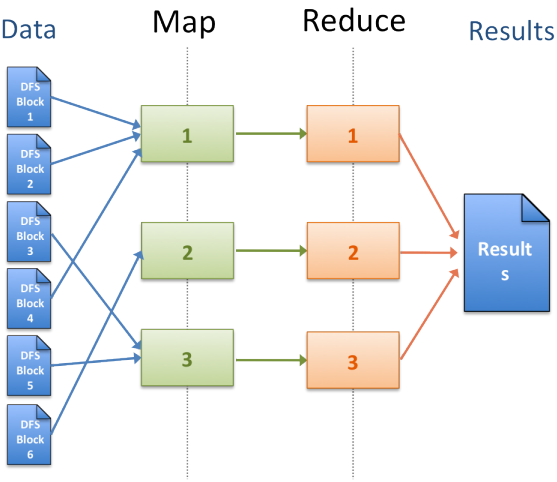
rr richrelevance®

Overview

Scaling a large database is not an overnight process, so it's difficult to plan and implement an upgrade that provisions for future traffic and usage without over-provisioning, which is very costly. One solution is a distributed database, which distributes its processing and storage load across multiple machines, and is therefore already designed for scalability. In contrast with a single database, a distributed database can be scaled in relatively short development cycles, so that its capacity is more precisely aligned with usage. When we considered approaches to scaling the RichRelevance systems, we evaluated several commercially available distributed databases. In the end we chose the Apache Hadoop open-source project. Though it presented a steep learning curve and required a fair amount of customization, we found that it offered the most power and flexibility.

Sample Hadoop Workflow

An example of what a Hadoop Cluster infrastructure diagram might look like.



Introduction: Solving for the Scalability Factor

Successful companies are faced with growing numbers of customers, and growing amounts of data to analyze and process. This is not a "problem" in the true sense of the word, but scaling a database and its underlying servers is neither inexpensive nor trivial, so many companies plan massive upgrades to expand processing capacity by two or three hundred percent, at regular intervals throughout the life of the company. Because database upgrades are planned far ahead of implementation, it's easy to over-provision or under-provision by varying degrees.

At RichRelevance, we *live* on customer data. We need to be able to capture and analyze *mountains* of customer data in real time, so that we can deliver the best recommendations to customers. But mountains are relatively static, so to switch metaphors, we need to analyze *oceans* of customer data, data that not only surges but is also constantly moving and changing. We never want to face the situation of having too much data to analyze quickly, but at the same time, we don't want be so over-provisioned that it taxes our resources, preventing us from taking the best care of our customers.

To address this problem, and find a solution that would allow us to easily scale in response to constantly changing customer demands,

RichRelevance turned to the Apache Hadoop open-source project for distributed data storage and processing. In this Speak Geek paper, we'll explore our use of this unique set of tools.

Pros and Cons of Distributed Computing

The RichRelevance team knew that one way around the "traditional" scaling model of upgrading a single database at regular intervals was to employ a distributed solution, such as a distributed database. In general, such solutions are easier to scale than single, monolithic databases because, by definition, the data already exists in multiple locations and does not require replication to secondary databases. Some distributed databases offer built-in redundancy and fault tolerance features. Others depend on the use of high availability storage and servers. The larger a cluster gets, the more often it experiences hardware failures, requiring extensive fault tolerance and resilience. Some distributed systems support heterogeneous hardware clusters—clusters made up of systems with different hardware types. Other systems require that every node in a cluster is nearly identical. Finally, distributed systems tend to offer excellent support for concurrent tasks.

Alternatively, the scaling problem could be solved by partitioning the data into several databases, but this approach has several problems: operational management of many independent DBs is expensive. Querying data across databases can be difficult. And perhaps the worst: this approach could simply trade off one large scalability problem for dozens or hundreds of smaller ones.

The Solution: the Elephant in the Server Farm

Apache Hadoop (named after the stuffed elephant of the original developer's son[1]) is not a distributed database, but rather a solution for distributed processing and data storage. RichRelevance turned to Hadoop because it had all of the advantages of a distributed database solution, while offering additional flexibility. For example, for reliability, Hadoop didn't require explicit pairing between nodes, which could result in data loss if two paired nodes were to go down. Beyond that, developers could specify replication levels for each data set. If warranted, a data set could be replicated 20 times across the cluster, for example.

RichRelevance turned to Hadoop because it had all the advantages of a distributed database solution while offering additional flexibility.

1 Vance, Ashlee. "Hadoop, a Free Software Program, Finds Uses Beyond Search" in The New York Times, March 16, 2009.

Also, RichRelevance found that many commercially distributed database solutions either had scalability limitations, such as additional complexities around adding nodes, or they would increase the price for every added node. They also found that many had limited ways to get data into the system. In addition, many were limited in their support for heterogeneous clusters; less powerful nodes would work with more powerful nodes, but would slow them down. Most importantly, RichRelevance felt that none of the available distributed database systems would ultimately solve its core problem, which was how to scale *data transformation*. RichRelevance's need for space was increasing exponentially. In April 2008, they were collecting roughly 200,000 events a day. A mere 18 months later, they were collecting close to a billion events a day.

The RichRelevance team knew that Hadoop, being an open-source solution, would require some heavy development time. However, the team did not see this as a major impediment, since the other options they considered, including distributed databases, would also require considerable customization to meet the team's needs. The team was also impressed that Hadoop had a growing list of well-established contributors for both research and production, including Adobe, Amazon, Facebook, IBM, LinkedIn, Twitter, and Yahoo!, and an active user community that could help answer questions and provide guidance.

Implementing Hadoop

As expected, the RichRelevance team experienced a steep learning curve in learning Hadoop. Hadoop is comprised of a variety of tools, or subprojects, such as those for getting data into Hadoop (see the sidebar for a full list of subprojects). The challenge was that each tool was in a different stage of development, and they overlapped in several areas, so it was difficult to know which was best for a given situation without extensive experimentation. Also, many basic functions, which are normally very straightforward using SQL, seemed unnecessarily complex and difficult using Hadoop's MapReduce process, for example.

However, after the initial learning period, the team felt that the effort paid off, and found that using the subproject "Pig" was actually much better suited than SQL for some tasks. Scott Carey, Performance Architect at RichRelevance said that "Although the easy stuff seemed

Apache Hadoop Subprojects

Currently (as of April, 2010) these include[1]

- **Avro:** A system that facilitates the serialization and de-serialization of data.
- **Chukwa:** A data collection system that operates on top of HDFS and MapReduce (See below).
- **HBase:** A large scale columnar database for use on top of HDFS.
- **Hadoop Distributed File System (HDFS):** Hadoop's own file distributed and redundant system.
- **Hive:** A SQL-like data warehouse infrastructure for use in Hadoop.
- **MapReduce:** A software framework for distributing processing across multiple nodes in a cluster configuration.
- **Pig:** A high-level language for developing Hadoop data analysis and transformation.
- **ZooKeeper:** A service for distributed application coordination.

1 http://hadoop.apache.org/

difficult, we soon learned that the hard stuff, when we were dealing with multiple input types, lots of filtering and aggregating, joining and grouping, was starting to seem really easy." While SQL is a *declarative* language, in that it emphasizes the end goal, Pig is a *procedural* language, which emphasizes the process. Using Pig, the RichRelevance team was able to troubleshoot parts of long queries with relative ease, a process that can be daunting using SQL.

The team found Pig to be more flexible than SQL. Pig can operate on semi-structured data stored in Hadoop, whereas SQL requires you to define and apply structure to your data before accessing it. Also, unlike SQL, Pig allows for nested data structures. Though SQL is very good at transactional operations on well-defined schemas, the team found that Pig operates best on bulk data in batch operations.

Once the RichRelevance team got over the initial learning curve, they also appreciated some of the finer nuances of how Hadoop differs from a distributed database. For example, the team often needed to build customized indexes from the data so it could be easily and efficiently consumed by outside applications. Using a database, this process involves numerous steps that include extracting the relevant data and storing it outside of the database. In contrast, the index-building process is much more natural using Hadoop, as it can occur without extracting and storing data. Hadoop allowed the team to build custom indexes with fewer network resources and considerably less coding.

Overtime, the team began expanding the Hadoop project to better meet its needs (as well as the potential needs of other users). Though Hadoop was best suited for large clusters of fairly weak machines, RichRelevance began working on code changes that would enhance Hadoop's ability to support smaller clusters of moderately powered machines. This involved optimizing the scheduling of tasks, allowing the stronger machines to take on more tasks, as well as customizing the MapReduce process to more efficiently distribute tasks across smaller numbers of machines.

Reaping the Rewards

As expected, the team found Hadoop to be much easier to scale than a traditional database. More importantly, the team found that not only could they scale Hadoop by a small amount, say a roughly 20% increase in capacity on an as needed basis, but they could scale in

a much more predictable manner than before. To scale by a given amount, the team knows how many servers to add for exactly what cost. Clearly, this made a strong impression on the business side of RichRelevance. They also found that Hadoop performed very well in heterogeneous environments including a mix of differently powered servers.

In addition, the team found benefits of using Hadoop that they didn't expect. For example, they knew that Hadoop would provide support for more concurrent tasks than a traditional database, but they were surprised by how much the system could take without showing a significant lag in performance.

In learning the tools, and experimenting with Hadoop, the team was able to "slam" Hadoop with multiple concurrent tasks with predictable impact. Needless to say, this atmosphere of experimentation was vital in the development process. The team was able to reliably run complex queries after a few hours of learning. For Albert Sunwoo, a Senior Engineer at RichRelevance, this was an "aha" moment: "I didn't have to query the database and wait along with a hundred other processes for the results to come back." The system showed very good support for ad hoc and automatic queries as well, which was very important to the RichRelevance team.

Hadoop is highly fault-tolerant. During one of the first hardware failures, the team was about to take action but found that the system had already self-healed and was back to working order without need for human intervention.

Finally, the team appreciated that, in addition to distributed processing, they could use the Hadoop cluster for storage. Unlike a database, the Hadoop cluster allows data spaces to grow very easily and offer built-in redundancy. It can also store any arbitrary, non-structured data as well as structured data, which was a welcome, additional feature that wasn't on the team's list of top requirements. Figure 1 illustrates how, over time, Hadoop reduced utilization of the database for storage.

Encouraged by their results in using Hadoop, RichRelevance plans to move more and more processing from legacy systems to Hadoop (see Figure 3).

FIGURE 1

Database Storage Utilization

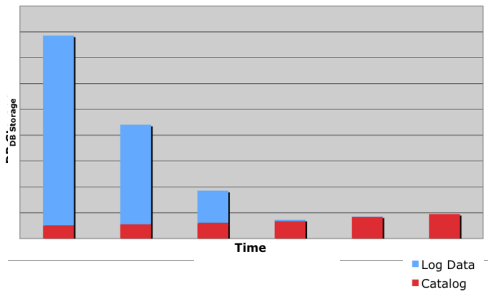
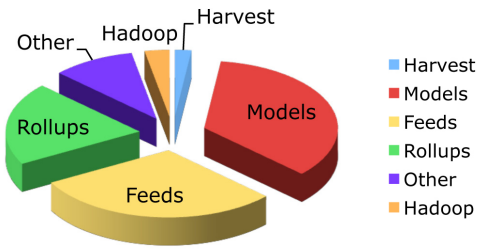


FIGURE 2

Before Hadoop Migration

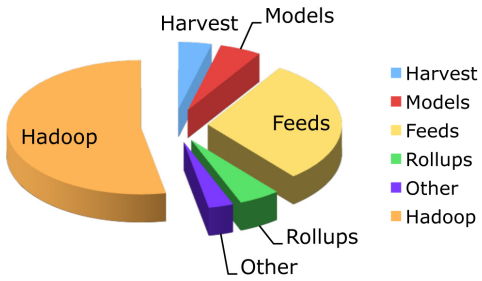


* Everything other than the Hadoop label above represents processing time on the database.

FIGURE 3

Processing Load after Migration

A large percentage of model building, rollups and other tasks have been migrated to Hadoop. Future growth will primarily occur in the Hadoop segment which is easier to scale.



* Everything other than the Hadoop label above represents processing time on the database.

Conclusion

After implementing Hadoop for a number of key projects, the RichRelevance team decided that though a single database might be best for a transactional environment, a toolbox like Hadoop is best for analytical environments, in which data can always be looked at differently, and can always be applied to different contexts. Though Hadoop required considerable customization, RichRelevance determined that it offered the most power, flexibility, reliability, and scalability compared with available alternatives. Hadoop enables RichRelevance to easily expand to accommodate more customers, more demand, better performance, and easily adapt to a rapidly changing customer environment.

Check www.richrelevance.com/speekgeek for the latest downloadable edition in this series.