21

DESIGNING DISTRIBUTED SYSTEMS: GOOGLE CASE STUDY

The ability to create an effective design is an important skill in distributed systems, requiring an awareness of the different technological choices featured throughout this book and a thorough understanding of the requirements of the relevant application domain. The eventual goal is to come up with a consistent distributed system architecture incorporating a consistent and complete set of design choices able to address the overall requirements. This is a demanding task and one that requires considerable experience with distributed systems development.

We illustrate distributed design through a substantial case study, examining in detail the design of the Google infrastructure, a platform and associated middleware that supports both Google search and a set of associated web services and applications including Google Apps. This includes the study of key underlying components including the physical infrastructure underpinning Google, the communication paradigms offered by the Google infrastructure and the associated core services for storage and computation.

Emphasis is placed on the key design principles behind the Google infrastructure and how they pervade the overall system architecture, resulting in a consistent and effective design.

21.1 Introduction

This book has focused on the key concepts that underpin the development of distributed systems, with an emphasis on addressing the main challenges of distributed systems, including heterogeneity, openness, security, scalability, failure handling, concurrency, transparency and quality of service. The subsequent treatment focuses inevitably on the constituent parts of a distributed system, including: communication paradigms such as remote invocation and its indirect alternatives; the programming abstractions offered by objects, components or web services; specific distributed systems services for security, naming and file system support; and algorithmic solutions such as coordination and agreement. It is equally important, however, to consider the overall design of distributed systems and how the constituent parts come together, addressing the inevitable trade-offs between the different challenges to derive an overall system architecture that meets the requirements of a large-scale application domain and operating environment. A fuller treatment of distributed systems design methods would necessarily take us into the field of software engineering methodologies for distributed systems. That is beyond the scope of this book; those interested should see the box below for some relevant topics and sources. Instead, we elect to provide insight into this area by presenting a case study of a complex distributed system, highlighting the key decisions and trade-offs involved in this design.

To motivate the studies in the book, Chapter 1 outlined three examples of key application domains that represent many of the major challenges in distributed systems: web search, massively multiplayer online games and financial trading. We could have picked any one of these areas and presented an enticing and illuminating case study, but we have chosen to focus on the first: web search (and indeed, we look beyond web search to more general support for web-based cloud services). In particular, in this chapter we present a case study on the distributed systems infrastructure that underpins Google (hereafter referred to as the Google infrastructure). Google is one of the largest distributed systems in use today, and the Google infrastructure has successfully dealt with a variety of demanding requirements, as discussed below. The underlying architecture and choice of concepts are also very interesting, picking up on many of the core topics presented in this book. A study of the Google infrastructure therefore

Software engineering for distributed systems

We refer the reader to the significant advances that have been made in areas such as:

- object-oriented design, including the use modelling notations such as UML [Booch *et al*. 2005];
- component-based software engineering (CBSE) and its relationship to enterprise architectures [Szyperski 2002];
- architectural patterns targeting distributed systems [Bushmann *et al*. 2007];
- model-driven engineering that seeks to generate complex systems (including distributed systems from higher-level abstractions (models) [France and Rumpe 2007].

provides a perfect way to round off our study of distributed systems. Note that, as well as providing an example of how to support web search, the Google infrastructure has emerged as a leading example of cloud computing, as will become apparent in the descriptions that follow.

Section 21.2 introduces the case study, providing background information on Google. Section 21.3 then presents the overall design of the Google infrastructure, considering both the underlying physical model and the associated system architecture. Section 21.4 examines the lowest level of the system architecture, the communication paradigms supported by the Google infrastructure; the subsequent two sections, Sections 21.5 and 21.6, discuss the core services provided by the Google infrastructure, featuring the services for storage and processing of massive quantities of data. Sections 21.3 to 21.6 together describe a complete middleware solution for web search and cloud computing. Finally, Section 21.7 summarizes the key points emerging from our discussion of the Google infrastructure. Throughout the presentation, an emphasis will be placed on identifying and justifying the core design decisions and the associated trade-offs that are inherent in the design.

21.2 Introducing the case study: Google

Google [www.google.com III] is a US-based corporation with its headquarters in Mountain View, California (the Googleplex), offering Internet search and broader web applications and earning revenue largely from advertising associated with such services. The name is a play on the word *googol*, the number 10^{100} (or 1 followed by a hundred zeros), emphasizing the sheer scale of information available in the Internet today. Google's mission is to tame this huge body of information: 'to organize the world's information and make it universally accessible and useful' [www.google.com III].

Google was born out of a research project at Stanford University, with the company launched in 1998. Since then, it has grown to have a dominant share of the Internet search market, largely due to the effectiveness of the underlying ranking algorithm used in its search engine (discussed further below). Significantly, Google has diversified, and as well as providing a search engine is now a major player in cloud computing.

From a distributed systems perspective, Google provides a fascinating case study with extremely demanding requirements, particularly in terms of scalability, reliability, performance and openness (see the discussion of these challenges in Section 1.5). For example, in terms of search, it is noteworthy that the underlying system has successfully scaled with the growth of the company from its initial production system in 1998 to dealing with over 88 billion queries a month by the end of 2010, that the main search engine has never experienced an outage in all that time and that users can expect query results in around 0.2 seconds [googleblog.blogspot.com I]. The case study we present here will examine the strategies and design decisions behind that success, and provide insight into design of complex distributed systems.

Before proceeding to the case study, though, it is instructive to look in more detail at the search engine and also at Google as a cloud provider.

The Google search engine • The role of the Google search engine is, as for any web search engine, to take a given query and return an ordered list of the most relevant results that match that query by searching the content of the Web. The challenges stem from the size of the Web and its rate of change, as well as the requirement to provide the most relevant results from the perspective of its users.

We provide a brief overview of the operation of Google search below; a fuller description of the operation of the Google search engine can be found in Langville and Meyer [2006]. As a running example, we consider how the search engine responds to the query 'distributed systems book'.

The underlying search engine consists of a set of services for crawling the Web and indexing and ranking the discovered pages, as discussed below.

Crawling: The task of the crawler is to locate and retrieve the contents of the Web and pass the contents onto the indexing subsystem. This is performed by a software service called Googlebot, which recursively reads a given web page, harvesting all the links from that web page and then scheduling further crawling operations for the harvested links (a technique known as *deep searching* that is highly effective in reaching practically all pages in the Web).

In the past, because of the size of the Web, crawling was generally performed once every few weeks. However, for certain web pages this was insufficient. For example, it is important for search engines to be able to report accurately on breaking news or changing share prices. Googlebot therefore took note of the change history of web pages and revisited frequently changing pages with a period roughly proportional to how often the pages change. With the introduction of Caffeine in 2010 [googleblog.blogspot.com II], Google has moved from a batch approach to a more continuous process of crawling intended to offer more freshness in terms of search results. Caffeine is built using a new infrastructure service called Percolator that supports the incremental updating of large datasets [Peng and Dabek 2010].

Indexing: While crawling is an important function in terms of being aware of the content of the Web, it does not really help us with our search for occurrences of 'distributed systems book'. To understand how this is processed, we need to have a closer look at indexing.

The role of indexing is to produce an index for the contents of the Web that is similar to an index at the back of a book, but on a much larger scale. More precisely, indexing produces what is known as an *inverted index* mapping words appearing in web pages and other textual web resources (including documents in *.pdf*, *.doc* and other formats) onto the positions where they occur in documents, including the precise position in the document and other relevant information such as the font size and capitalization (which is used to determine importance, as will be seen below). The index is also sorted to support efficient queries for words against locations.

As well as maintaining an index of words, the Google search engine also maintains an index of *links*, keeping track of which pages link to a given site. This is used by the PageRank algorithm, as discussed below.

Let us return to our example query. This inverted index will allow us to discover web pages that include the search terms 'distributed', 'systems' and 'book' and, by careful analysis, we will be able to discover pages that include all of these terms. For example, the search engine will be able to identify that the three terms can all be found in amazon.com, www.cdk5.net and indeed many other web sites. Using the index, it is therefore possible to narrow down the set of candidate web pages from billions to perhaps tens of thousands, depending on the level of discrimination in the keywords chosen.

Ranking: The problem with indexing on its own is that it provides no information about the relative importance of the web pages containing a particular set of keywords – yet this is crucial in determining the potential relevance of a given page. All modern search engines therefore place significant emphasis on a system of ranking whereby a higher rank is an indication of the importance of a page and it is used to ensure that important pages are returned nearer to the top of the list of results than lower-ranked pages. As mentioned above, much of the success of Google can be traced back to the effectiveness of its ranking algorithm, *PageRank* [Langville and Meyer 2006].

PageRank is inspired by the system of ranking academic papers based on citation analysis. In the academic world, a paper is viewed as important if it has a lot of citations by other academics in the field. Similarly, in PageRank, a page will be viewed as important if it is linked to by a large number of other pages (using the link data mentioned above). PageRank also goes beyond simple 'citation' analysis by looking at the importance of the sites that contain links to a given page. For example, a link from *bbc.co.uk* will be viewed as more important than a link from Gordon Blair's personal web page.

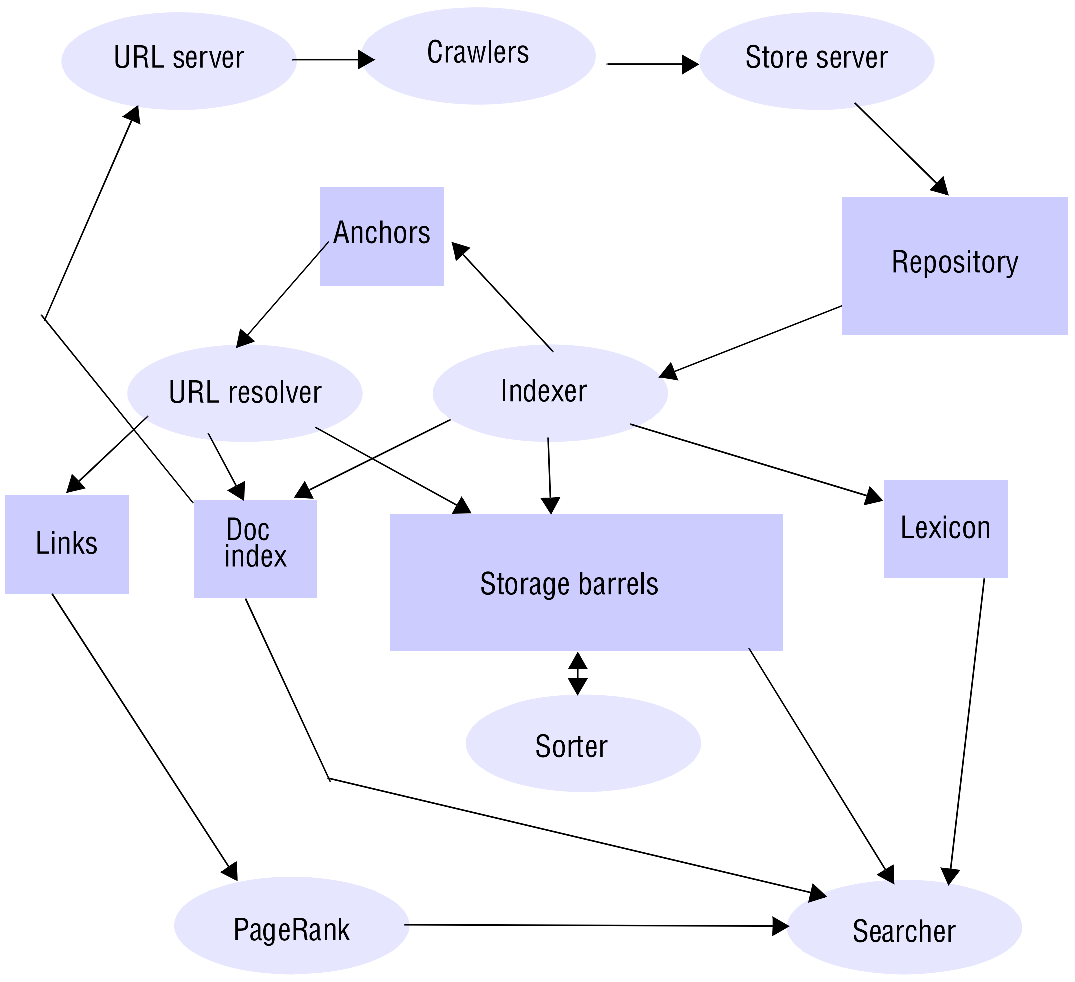
Ranking in Google also takes a number of other factors into account, including the proximity of keywords on a page and whether they are in a large font or are capitalized (based on the information stored in the inverted index).

Returning to our example, after performing an index lookup for each of the three words in the query, the search function ranks all the resulting page references according to perceived importance. For example, the ranking will pick out certain page references under amazon.com and www.cdk5.net because of the large number of links to those pages from other 'important' sites. The ranking will also prioritize pages where the terms 'distributed', 'systems' and 'book' appear in close proximity. Similarly, the ranking should pull out pages where the words appear near the start of the page or in capitals, perhaps indicating a list of distributed systems textbooks. The end result should be a ranked list of results where the entries at the top are the most important results.

Anatomy of a search engine: The founders of Google, Sergey Brin and Larry Page, wrote a seminal paper on the 'anatomy' of the Google search engine in 1998 [Brin and Page 1998], providing interesting insights into how their search engine was implemented. The overall architecture described in this paper is illustrated in Figure 21.1, redrawn from the original. In this diagram, we distinguish between services directly supporting web search, drawn as ovals, and the underlying storage infrastructure components, illustrated as rectangles.

While it is not the purpose of this chapter to present this architecture in detail, a brief overview will aid comparison with the more sophisticated Google infrastructure available today. The core function of *crawling* was described above. This takes as input lists of URLs to be fetched, provided by the *URL server*, with the resultant fetched pages placed into the *store server*. This data is then compressed and placed in the *repository* for further analysis, in particular creating the index for searching. The indexing function is performed in two stages. Firstly, the *indexer* uncompresses the data in the repository

Figure 21.1 Outline architecture of the original Google search engine [Brin and Page 1998]



and produces a set of *hits*, where a hit is represented by the document ID, the word, the position in the document and other information such as word size and capitalization. This data is then stored in a set of *barrels*, a key storage element in the initial architecture. This information is sorted by the document ID. The *sorter* then takes this data and sorts it by word ID to produce the necessary inverted index (as discussed above). The indexer also performs two other crucial functions as it parses the data: it extracts information about links in documents storing this information in an *anchors* file, and it produces a *lexicon* for the analyzed data (which at the time the initial architecture was used,consisted of 14 million words). The anchors file is processed by a *URL resolver*, which performs a number of functions on this data including resolving relative URLs into absolute URLs before producing a *links* database, as an important input into *PageRank* calculations. The URL resolver also create a *doc index*, which provides input to the URL server in terms of further pages to crawl. Finally, the *searcher* implements the core Google search capability, taking input from the doc index, PageRank, the inverted index held in the barrels and also the lexicon.

One thing that is striking about this architecture is that, while specific details of the architecture have changed, the key services supporting web search – that is, crawling, indexing (including sorting) and ranking (through PageRank) – remain the same.

Figure 21.2 Example Google applications

Application	*Description*
Gmail	Mail system with messages hosted by Google but desktop-like message management.
Google Docs	Web-based office suite supporting shared editing of documents held on Google servers.
Google Sites	Wiki-like web sites with shared editing facilities.
Google Talk	Supports instant text messaging and Voice over IP.
Google Calendar	Web-based calendar with all data hosted on Google servers.
Google Wave	Collaboration tool integrating email, instant messaging, wikis and social networks.
Google News	Fully automated news aggregator site.
Google Maps	Scalable web-based world map including high-resolution imagery and unlimited user-generated overlays.
Google Earth	Scalable near-3D view of the globe with unlimited user-generated overlays.
Google App Engine	Google distributed infrastructure made available to outside parties as a service (platform as a service).

Equally striking is that, as will become apparent below, the infrastructure has changed dramatically from the early attempts to identify an architecture for web search to the sophisticated distributed systems support provided today, both in terms of identifying more reusable building blocks for communication, storage and processing and in terms of generalizing the architecture beyond search.

Google as a cloud provider • Google has diversified significantly beyond search and now offers a wide range of web-based applications, including the set of applications promoted as Google Apps [www.google.com I]. More generally, Google is now a major player in the area of cloud computing. Recall that cloud computing was introduced in Chapter 1 and defined as 'a set of Internet-based application, storage and computing services sufficient to support most users' needs, thus enabling them to largely or totally dispense with local data storage and application software'. This is exactly what Google now strives to offer, in particular with significant offerings in the *software as a service* and *platform as a service* areas (as introduced in Section 7.7.1). We look at each area in turn below.

Software as a service: This area is concerned with offering application-level software over the Internet as web applications. A prime example is Google Apps, a set of web-based applications including Gmail, Google Docs, Google Sites, Google Talk and Google Calendar. Google's aim is to replace traditional office suites with applications supporting shared document preparation, online calendars, and a range of collaboration tools supporting email, wikis, Voice over IP and instant messaging.

Several other innovative web-based applications have recently been developed; these and the original Google Apps are summarized in Figure 21.2. One of the key observations for the purposes of this chapter is that Google encourages an open approach to innovation within the organization, and hence new applications are emerging all the time. This places particular demands on the underlying distributed systems infrastructure, a point that is revisited in Section 21.3.2.

Platform as a service: This area is concerned with offering distributed system APIs as services across the Internet, with these APIs used to support the development and hosting of web applications (note that the use of the term 'platform' in this context is unfortunately inconsistent with the way it is used elsewhere in this book, where it refers to the hardware and operating system level). With the launch of the Google App Engine, Google went beyond software as a service and now offers its distributed systems infrastructure as discussed throughout this chapter as a cloud service. More specifically, the Google business is already predicated on using this cloud infrastructure internally to support all its applications and services, including its web search engine. The Google App Engine now provides external access to a part of this infrastructure, allowing other organizations to run their own web applications on the Google platform.

We will see further details of the Google infrastructure as this chapter unfolds; refer to the Google web site for further details of the Google App Engine [code.google.com IV].

21.3 Overall architecture and design philosophy

This section looks at the overall architecture of the Google system, examining:

- the physical architecture adopted by Google;
- the associated system architecture that offers common services to the Internet search engine and the many web applications offered by Google.

21.3.1 Physical model

The key philosophy of Google in terms of physical infrastructure is to use very large numbers of commodity PCs to produce a cost-effective environment for distributed storage and computation. Purchasing decisions are based on obtaining the best performance per dollar rather than absolute performance with a typical spend on a single PC unit of around $1,000. A given PC will typically have around 2 terabytes of disk storage and around 16 gigabytes of DRAM (dynamic random access memory) and run a cut-down version of the Linux kernel. (This philosophy of building systems from commodity PCs reflects the early days of the original research project, when Sergey Brin and Larry Page built the first Google search engine from spare hardware scavenged from around the lab at Stanford University.)

In electing to go down the route of commodity PCs, Google has recognized that parts of its infrastructure will fail and hence, as we will see below, has designed the infrastructure using a range of strategies to tolerate such failures. Hennessy and Patterson [2006] report the following failure characteristics for Google:

- By far the most common source of failure is due to software, with about 20 machines needing to be rebooted per day due to software failures. (Interestingly, the rebooting process is entirely manual.)