On centralized and decentralized distributed systems

Stephan Soller, Computer Science and Media
Stuttgart Media University
ss312@hdm-stuttgart.de

Abstract

Recent distributed systems needed a great amount of research to scale. Older distributed systems achieved global scale seemingly without this much effort. This paper takes a look at the history of several globally distributed systems (mail, DNS, HTTP, Google, Facebook and Twitter) and the differences in their initial designs and how these relate to scalability.

It is found that distribution of operations and authority provides an horizontal scalability layer. But probably at the cost of difficult coordination and slower development.

1. Introduction

During recent years scalability was a major focus of distributed systems. Through global systems like Google search or Facebook much research has been conducted. Many different methods have been tried to sustain the sometimes exponential growth of these platforms. Systems that failed to scale were often deserted by their users who chose to use more responsive systems of the competition. One popular example of such a failed system is Friendster [1].

Yet seemingly unnoticed in all these recent developments some global systems have been around longer and achieved similar scales. The most obvious one is the email system since pretty much any registration (e.g. at Facebook) requires the user to have a valid mail box. Others are the domain name service (DNS) and even the HTTP based internet itself as most users perceive it today: A network of websites running on many different webservers.

These older systems are the basic infrastructure used by todays modern global systems. Yet the scalability of the HTTP based internet, mail or DNS does not seem to require the intense amount of research and effort put into Google or Facebook.

This paper tries to explore the difference in design between some older (mail, DNS, HTTP) and newer globally distributed systems (Google, Facebook, Twitter). To find the key differences between them and how they relate to scalability.

2. Google, Facebook and Twitter

This section provides a short history of some systems that have been the recent focus of some major scalability research: Google, Facebook and Twitter. It doesn't explain what they are or how they operate but instead tries to shed some light on the circumstances and technical mindset that lead to their creations.

Google

Around 1996 Google started out as a centralized indexing architecture build for improved search quality, scalability and academic research [2] [3]. Coming from the Stanford University it had a strong academic mindset of understanding and solving the challenges at hand. To scale alongside the growing web was an intrinsic aspect of its system design.

At Google technologies like GFS [4] and MapReduce [5] were created, each with scalability at heart. They were the base that allowed Google to efficiently handle the vast amounts of data and requests. Systems at Google are running in a closed and optimized environment and to the user they are accessible as a normal website. From the users perspective the distribution is practically invisible.

Facebook

After about one month of development by Mark Zuckerberg Facebook was released in 2004 [6]. At first as a Harvard-only social network but it was subsequently opened to other US universities, high schools and later on everyone. The expansion was carried out step by step.

Before each expansion additional capacity was added to sustain the expected growth of the user base [7].

The first version of Facebook was a website build with the open source LAMP stack. Over time parts of that stack were replaced with custom build technologies like Cassandra and the HipHop virtual machine for PHP. Permanent changes and improvements like these were necessary to sustain Facebooks growth and keep the website usable. While Facebook itself is operated in a closed environment most of their software is either open source or custom build and subsequently released as open source.

Twitter

In 2006 Twitter was released to the public. Initially it was meant to be an SMS based internal company information exchange [8]. It quickly changed into a Ruby on Rails based website using MySQL for storage. With this monolithic code base scalability was mostly achieved by "throwing machines at the problem". But further optimization and development became more difficult. As a consequence large parts of the infrastructure were redesigned, mostly as distributed services on top of the JVM [9].

Twitter too operates its infrastructure in a closed environment but also heavily uses and contributes to open source software [10].

Similarities

Although these systems do quite different things there are some common themes in how they came to be and in how they are operated.

In case of Facebook and Twitter (and many others) they started out as normal websites. After attracting many users the focus on scalability and later on distribution was simply a means to keep these websites running. Google differs here because it was specifically designed for the scale and growth of the internet. This might be attributed to its strong academic roots.

In all three cases distribution was primarily used to maintain proper performance at the scales in question. For the users themselves the distribution is completely invisible. All systems are pretty much perceived as websites or web applications.

Further similarities can be found in the way these systems are managed and operated: Usually the entire infrastructure is owned and operated by the respective company. The company forms one central instance to make decisions and to coordinate different parts of the infrastructure or different development teams. This allows to coordinate even radical changes efficiently. For example Twitters migration to the JVM.

On the operations side pretty much the entire infrastructure runs as a tightly supervised closed system. APIs to interface with e.g. Google are provided by they don't allow direct access to parts of the system, for example files on one clusters GFS. From the outside these systems are effectively black boxes. This allows radical changes without breaking any outwards compatibility. Having the entire infrastructure in one hand also allows to optimize low-level aspects. For example details of the Linux kernel network stack or the physical distance and arrangement of servers in the racks.

There are exceptions to this. For example Dropbox uses Amazon S3 to store user data instead of operating their own storage system. But this is invisible from the outside and can possibly change.

In regards to technology all companies publish or share some of their key technologies as open source. For example many companies use Hadoop which is based on the Google File System and MapReduce papers published by Google. In turn Google probably uses software published by other companies or at least draws inspiration from them. This creates to an open development culture focused on overcoming challenges instead of keeping solutions hidden from the competition.

3. Mail, DNS and HTTP

These systems have been around since the beginning of the internet. To some extend they have even been the reasons for it's creations or at least accelerated it. Again this section doesn't try to describe how they work but what the mindset was when they came to be.

Mail

Development on mail systems began in the 1960s at multiple projects. Many big computer systems developed their own way to send messages between users. This wasn't a "planned" feature but some users discovered the need for sending messages and others implemented programs and commands for it. At that time message transmission was only possible between users working on the same mainframe (e.g. MITs CTSS) [11].

With the creation of the ARPANet many of these computers became connected with each other. Networked mail delivery was one of the first proposed applications for the ARPANet and evolved in the 1970s. As the globally connected internet was yet to come much effort went into getting data across the different systems and network fragments (e.g. by putting routing paths into the recipients address). Development happened in a very collaborative way and information and ideas were shared and discussed openly. In fact the RFC (Request for Comments) infrastructure emerged as part of the ARPANet project.

Since the ARPANet was small and there were more pressing matters (e.g. message routing) it seems not much effort was invested into scalability of the system. It grew as disconnected islands which then were merged by the ARPANet. Because of that there never was a central mailbox repository where all the traffic hit. RFC 882 even states that "mail system implementers long ago recognized the impossibility of centralizing mailbox names" [12] which is more or less what Twitter does today.

During its long history some parts of the mail system were replaced or obsoleted (e.g. routing paths in addresses). But most early efforts seem to focus on common data formats and establishing connections between systems. This in part lead to the development of DNS.

Later on private providers started to handle most of the mail traffic and their mail servers needed to scale with the growing demand. This performance oriented scaling does not seem to require the extreme amount and effort scalability of Google or Facebook required [13].

Maybe performance and efficiency was a given requirement at the time and not noteworthy. After all compared with todays computers the mainframes of that era had next to no capacity.

Domain Name System

The history of DNS starts with the ARPANet around the 1970s. At first the ARPANet used a centrally maintained hosts.txt file to resolve host names to IP addresses. This file was updated nightly and then copied to all systems connected to the ARPANet. Since the ARPANet was growing quickly it didn't take long until maintenance and distribution of the hosts.txt file became impractical.

DNS was designed to replace the hosts.txt file [14] but also had the purpose to simplify mail routing and delivery. Two major factors in its distributed design were scalability and zones of authority [15]. This distributed not only the lookup of domain names but also the maintenance of the DNS database (or it's individual zones) across the network.

During the decades of operation several aspects of DNS were extended. Incremental Zone Transfer (IXFR), NOTIFY, dynamic updates and other extensions each solved previous shortcomings of the system [16]. It should also be noted that politics played a significant role in DNS as the internet in general moved towards commercialization.

HTTP

HTTP (or the web in general) was conceived around 1990 to allow CERNs researchers to better coordinate and document projects. Or simply to make the information at CERN accessible to the people who needed them. The idea was to overcome limits of hierarchical systems like newsgroups and to freely structure and link information. To create a "web" of linked hypertext documents. Much like a wiki of today.

It was decentralized so that no central control or coordination is required to link systems together. Heterogeneity also was a design goal to allow different client "browser" programs to interact with different "hypertext servers". This was necessary because people at CERN worked with many different systems (VM/CMS, Macintosh, VAX/VMS, Unix) and for each of those a client had to be written. On the server side many different databases were proposed to be accessible via "hypertext servers", for example newsgroups, CERNDOC, filesystems, the telephone book and even the unix manual. Each one serving its information as hypertext [17].

The Hypertext Transfer Protocol (HTTP) was designed as the glue between such clients and servers. Its structure (a header and body) and the headers themselves were heavily inspired by the mail format. Scalability it seems wasn't an explicit concern during HTTPs initial design. It's not mentioned in the early design documents or RFCs. However the protocol was stateless and idempotent on purpose [18], probably to allow better caching. Later on HTTP was revised by deriving and following the REST architecture [19]. Increasing scalability and even

tolerating "anarchic scalability" was one of the goals of this effort and lead to HTTP 1.1.

Similarities

Mail, DNS and HTTP all have different purposes but yet again there are some similarities.

Servers of all three systems are operated by many different instances and companies. Not one systems operation is in the hands of a single instance. DNS has a central administrative body though. Which servers are operated by whom is usually closely related to real world organizational structures or trust zones. With DNS and HTTP this was probably the obvious thing to do at the time. Whoever wants to operate servers with global names or publish documents also runs the servers to do so. The mail system pretty much started out with this structure since it emerged from the mainframes of different organizations.

Development however is coordinated through one more or a less central instance: the IETF. There problems, ideas and solution are discussed openly to foster further development.

Maybe as a consequence all systems are build for interoperability, at least DNS and HTTP. The mail system emerged out of many different implementations and was standardized to be interoperable. There wasn't much competition between "platforms" but different implementations were gradually assimilated by continuously revising the infrastructure.

From the clients point of view no system has a "central interface". Instead the client directly interfaces with the required servers. In day to day usage however this fact is rarely if at all noticed. For users there isn't much of a difference between sharing a user name or a mail address.

4. Key differences in design

The previously discussed distributed systems all have some differences and similarities with one another. This part of the paper highlights some of the key differences between older and newer distributed systems.

Central gateway

Google, Facebook and Twitter have a centralized interface to the rest of the internet: Their respective websites. Components implementing this central "gateway" must be able to handle the entire amount of data and scale with the growth of the entire system. The load on these gateways can be distributed via DNS and HTTP load balancing but semantically its still one point where all the traffic comes through.

Mail, DNS and HTTP on the other hand lack this central point. Instead the address of whatever you want to interact with already contains enough information to figure out which server to contact (e.g. a mail address, domain name or URL). Users sharing their addresses already form a simple distributed directory.

User names at Google, Facebook and Twitter don't have that property. When scaled to a distributed system a kind of directory lookup or distributed hashing is required to get the location of the object of interest. Again this central component must be able to cope and scale with the total load of the entire system.

Centralized operations

With centralized operations the entire infrastructure can be optimized, down to every low-level aspect. From the hardware of the different servers to the physical layout and structure of the internal network. Even entire data centers can be optimized, e.g. for energy efficiency. Every aspect of the software stack can be optimized too. Using efficient operating systems, making the operating system more efficient or using specialized deployment and monitoring tools. Changes affecting the entire system can be deployed efficiently.

These are important aspects since they make it easier to quickly identify and solve problems and bottlenecks at every layer. This in turn opens up more potential to improve the scalability of every component and the entirety of the system.

Its difficult to say if decentralized systems could have handled the rapid growth some platforms experienced (e.g. Facebook). Maybe it would have taken to long to identify and solve problems of the basic decentralized interactions. In the worst case users would've lose interest in the new technology.

However once basic interactions of a decentralized system work the scalability depends only on the performance required by each node. It's no longer dependent on the size of the entire system. If a company doesn't cause much load on their node they don't need to put much effort into scalability. If the infrastructure of one node is operated by a single instance and scalability is

needed all the optimizations from above can be done here too. In that scheme Facebook is just one insanely scaled up node in the HTTP network. Others without these scalability demands are not affected by the optimizations done by Facebook.

Decentralized operations also make the system more resilient. Different parts can be operated at different locations around the world by different legal entities. Thus they're better protected from natural disasters and legal turmoil. Large cooperations like Google can do the same but still provide one legal attack point: The host cooperation. In contrast companies that operate e.g. different mail servers can be legally independent of each other.

Centralized authority and management

The impact of authority and management on scalability is not clear.

When authority is centralized one company alone has authority over the system and how it is used. This can be used to filter out unwanted or inappropriate content, for example pornographic content, troll comments or malicious apps in an app store. That in turn maybe useful or even necessary to uphold a certain image or purpose of a product.

But as soon as user generated content becomes important things get more problematic. Generally users assume to have authority over their own content (e.g. images or comments). The legal situation however is complicated. In extreme cases this becomes a conflict of interest between the users freedom of expression and the interests of the company. The matter gets even more complicated when the users of a system span multiple cultural zones and what is regarded as inappropriate content can differ between them. When one company holds central authority over a system it has to take care of and is responsible for all of this. While there is no obvious technical impact on scalability it requires *the company* to scale and adapt to different cultures and laws.

With decentralized authority the above burden is distributed between all the different companies or instances that operate the parts of the system. This allows the system to adapt and match the cultural standards of whoever is operating the part. Its also much more in line with current territorial based legislation. At least if a server e.g. hosting German content is also operated in Germany.

Differences in development

All the systems discussed in this paper use a quite free development culture. But there are some subtle differences that might have an impact on the scalability a system achieves.

All use centralized instances to coordinate development. Either a company or some form of committee or work group. At companies like Google small teams develop and publish "working and battle proven prototypes" as open source projects. These are internally consistent and optimized for their purpose.

Communication and coordination in a small team is more direct, efficient and productive than in a work group or committee. Especially when the team can work full time on the project whereas members of work groups often do their work on the sidelines. In a committee or working group communication is probably more difficult and slow. In turn it becomes difficult or takes longer to create consistent and optimized systems.

Also interest of different committee members probably differ from one another more than the interests of different team members. This gives rise to political dynamics that can slow down development or lead to inconsistent software.

Fast growing systems sometimes need to change quickly to stay scalable. With working groups or committees this is more difficult than with small teams.

5. Popularity of centralized systems

There are probably reasons why more recent distributed systems are often centralized. That is they're basically websites that have been scaled up. While an authoritative search for these reasons is well beyond the scope here the above information allows us to speculate about them. Even if only briefly.

Facebook and Twitter started out as centralized websites. While monetization maybe a reason why they stay centralized its probably not the reason for the initial centralized design. At first Facebook, Twitter and neither Google had a clear plan on how to earn money with their services. They were just useful to people.

Surveillance probably wasn't a reason either: In case of Google and Facebook government agencies probably

reacted to the new communication trends and requested and implemented new surveillance methods.

Maybe after 2000 many developers simply grew up with HTTP. Whenever something needed to be done "creating a websites" simply was the first way to do it. In that case future development will probably erode the abstractions and constraints set by HTTP. Down until more universal semantics are achieved that are usually provided by TCP (bidirectional channel) and UDP (low latency). Recent development into websockets seem to hint into this direction. But much more research is needed before this chain of conclusions can be seriously considered.

However when comparing the implementation of a centralized and decentralized system one fact becomes obvious: A website as central interface to a service is just *much more simple*.

Decentralization on the network level has become increasingly difficult and messy. Workarounds like NAT prevent simple establishment of end to end connections. Insecure software lead to widespread deployment of restrictive firewalls and routers. Each with their own quirks that need to be taken into account and circumvented (e.g. UPnP and firewall auto-configuration). Some vendors are unwilling or extremely slow to take up new technologies. For example the entire IPv6 migration or nearer to the software side of things: SCTP support on Windows. All this combined makes the basic infrastructure for a decentralized system (e.g. peer to peer) ridiculously more complex than a centralized one.

6. Conclusion

Given all the above information it can be assumed that decentralized systems are probably *simpler* to scale once the basic infrastructure is running. If it needs more or less effort than a centralized system is difficult to tell though.

That older distributed systems didn't need much effort to achieve scalability was a misconception of the author. DNS and later on HTTP too received a fair amount of scalability research. The mail system is somewhat special here since it started out in a decentralized and disconnected world. And at that time it wasn't possible to centralize it. DNS on the other hand provides a different example: It started out as a centralized list and with deliberate effort became decentralized. The misconception was probably caused by the current popularity of Google, Facebook & Co. in scalability research. They are the current "rock stars" in this area while the mail system, DNS and others tag along just fine.

However it became clear that decentralized systems have some intrinsic advantages. They don't have a central "gateway" and if the address contains enough information a centralized directory isn't needed either. In centralized systems these two components have to scale with the load of the entire system. Decentralization of operations and authority also provide an resilient horizontal scalability layer. The cost and effort is automatically spread among many organizations. Distributing authority allows a system to naturally match the organizational and legal structures of its users.

Developing the basic infrastructure of a decentralized systems however is much more difficult and on the long run it's probably more difficult to adapt and change. If a systems operations are centralized its easier to extensively optimize and can be changed faster.

7. Further thoughts

It would also be interesting to compare the number of changes in different distributed systems. For example the number of major technological overhauls or redesigns in Google, Twitter, Facebook, mail, DNS and HTTP. This should give a more accurate impression of how much effort was actually invested into the scalability of each system. For older systems the data is readily available in the form of RFCs. Closed systems however don't publish the details of all changes and this would make it difficult to obtain comparable data.

It might also be worthwhile to include more recent decentralized systems like XMPP and torrent networks in the analysis. It's likely they have been influenced by recent centralized systems. How did they came to be and how they relate to other distributed systems might offer a glimpse into future distributed systems.

Centralized distributed systems tend to become decentralized internally. This allows easier parallel development of components, maintenance and fault tolerance. In that they mimic the structures of decentralized systems, yet they retain some kind of global control and coordination over the entire infrastructure. This instance allows to incrementally optimize all components or even replace entire parts of the infrastructure if necessary.

Due to the lack of a sufficiently powerful "optimizing instance" older distributed systems probably respond more slowly to change, if at all (e.g. mail). It would be an very interesting research topic if the advantages of centralized and decentralized distributed systems can be combined or not. Decentralized distributed systems lack a central organization or "owner" and maybe this is mutually exclusive to having a powerful enough "optimizing instance" for the entire system.

8. References

[1] Friendster founder on the rise and fall of America's first big social network
http://latimesblogs.latimes.com/technology/2009/07/friendster.html
Retrieved 2014-01-31

[2] Google.com - Our history in depth
http://www.google.com/about/company/history/
Retrieved 2014-01-31

[3] The Anatomy of a Large-Scale Hypertextual Web Search Engine
http://infolab.stanford.edu/~backrub/google.html
Retrieved 2014-01-31

[4] The Google File System
http://research.google.com/archive/gfs-sosp2003.pdf
Retrieved 2014-01-31

[5] MapReduce: Simplified Data Processing on Large Clusters
http://research.google.com/archive/mapreduce.html
Retrieved 2014-01-31

[6] Did Mark Zuckerberg's Inspiration for Facebook Come Before Harvard?
http://readwrite.com/2009/05/10/mark_zuckerberg_inspiration_for_facebook_before_harvard
Retrieved 2014-01-31

[7] The Technical Architecture Behind Facebook
hhttp://frrl.wordpress.com/2011/03/22/the-technical-architecture-behind-facebook/
Retrieved 2014-01-31

[8] The Real History of Twitter, In Brief
http://twitter.about.com/od/Twitter-Basics/a/The-Real-History-Of-Twitter-In-Brief.htm
Retrieved 2014-01-31

[9] New Tweets per second record, and how!
https://blog.twitter.com/2013/new-tweets-per-second-record-and-how
Retrieved 2014-01-31

[10] Twitter - Open Source
https://about.twitter.com/company/open-source
Retrieved 2014-02-02

[11] The History of Electronic Mail, Tom Van Vleck
http://www.multicians.org/thvv/mail-history.html
Retrieved 2014-02-02

[12] RFC 882 - DOMAIN NAMES - CONCEPTS and FACILITIES
https://www.ietf.org/rfc/rfc0882.txt
Retrieved 2014-02-02

[13] A history of e-mail: Collaboration, innovation and the birth of a system, Dave Crocker
http://www.washingtonpost.com/national/on-innovations/a-history-of-e-mail-collaboration-innovation-and-the-birth-of-a-system/2012/03/19/gIQAOeFEPS_story.html
Retrieved 2014-02-02

[14] Short History of DNS
http://support.easystreet.com/domain-name-services/97-short-history-of-dns
Retrieved 2014-02-02

[15] One History of DNS, Ross Wm. Rader
http://www.byte.org/blog/one-history-of-dns.pdf
Retrieved 2014-02-02

[16] The Evolution of DNS – Starting from the Beginning
http://blog.neustar.biz/dns-matters/the-evolution-of-dns-starting-from-the-beginning/
Retrieved 2014-02-02

[17] Information Management: A Proposal, Tim Berners-Lee
http://www.w3.org/History/1989/proposal.html
Retrieved 2014-02-02

[18] HyperText Transfer Protocol Design Issues
http://www.w3.org/Protocols/DesignIssues.html
Retrieved 2014-02-02

[19] Architectural Styles and the Design of Network-based Software Architectures
http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf
Retrieved 2014-02-02