Google: The Software Giant   
Do Ultra-Large Scale Software Systems Still Follow Accepted Hints For System Design?

Eric James Rapos

# Introduction

In the ever growing realm of software systems, a newly emerging term has come to fruition; the notion of an ultra-large scale software system, or ULSS for short. Systems are rapidly increasing in size and scope, and potentially require a different paradigm to deal with them. The definition of a ULSS is something that is currently rather fluid, and determining if a system truly is ultra-large is somewhat subjective; however there are a few characteristics that help us describe a system that is, or may soon be considered to be, a ULSS [1].

One of the main characteristics is simply scale; the increase in a number of attributes will tend to hint at a system’s classification as ultra large. These attributes include, but are not limited to: lines of code, users, amount of data, number of connections, and the number of hardware elements. These are merely high level indicators, and there are a number of more accurate descriptors of ultra-large scale software systems. A ULSS is typically decentralized, meaning that there is no central control and no central data storage. A ULSS is also accepted to be inherently conflicting and containing diverse requirements; due to the numerous possible uses, each user may want to interact with the system in a different way, some of which may even conflict with others. As with all systems, but even more so in a ULSS, there is continuous development and evolution, adding new features, expanding functionality and generally increasing the scope of the system. Additionally, it is important to note that in a ULSS, the components are rarely consistent and always evolving to suit the needs of the users; parts of the system may swap out from time to time, and may be inconsistent with other elements, but are still required to function. Another interesting characteristic is the notion that the people involved in a ULSS may no longer be just the users, but a part of the system, impacting its functionality; thus removing the hard line between user and system. Failures are a regular part of all software systems, but when considering a ULSS which is by definition much larger than any average system, these failures occur more often, and may be more severe. It is a characteristic of a ULSS that these errors will occur, and must be accounted for. Finally, the paradigms for the acquisition and policies of a ULSS differ greatly from the existing paradigms for large systems, in that the acquisition of a ULSS will be simultaneous with the operation of the system.

The question this paper aims to answer is that with this new classification of ultra-large scale software systems in place, does current software, especially those possibly classified as a ULSS still follow widely accepted design hints for software that have existed for almost thirty years? Lampson presents a number of design hints for software [8], and using Google as an example ULSS, the intent is to demonstrate that a ULSS does indeed follow these outlined hints.

# Google: Ultra-Large?

Before proceeding into the design hints presented by Lampson, it is important highlight the characteristics of a ULSS demonstrated by Google to indicate that it is (or rather could be considered) a ULSS.

Let us first look at the scale of Google as a software system. While the number of lines of code within the entirety of the Google system not widely available, it is a safe bet to be in the hundreds of millions, and not quite nearing a billion. This is a rather large number and can certainly fall into the ultra-large scope. Additionally, looking at the number of users, Google reports that on any given day, there are 620 million visitors to google.com, with an average of 7.2 billion page views [2]; this means that the number of users could certainly be within the realm of ultra-large. In terms of data, Google processes 20 petabytes of data on average each day [2]. However, beyond simply scale Google demonstrates a number of the other characteristics of a ULSS. Google is certainly not a centralized system in any means; in addition to being a system of systems with different control centers, Google’s servers are largely decentralized. There are Google data centers all over the world [3] that are used to help the software giant operate the way it needs to provide services to its users. The continuous evolution of Google is another demonstration of a ULSS characteristic, in that Google is always evolving, and adding new features while the system remains running; a system like Google could not shut down to add a new feature to search, or to update a part of YouTube; downtime for Google is not an option. This is something that leads to another characteristic of a ULSS demonstrated by Google, and that is the anticipation of normal failures; Google has in place a number of different techniques, including their implementation of data centers for redundant storage, that prevent any sort of down time in the event of a disaster/failure of a storage device.

Based on these claims, it is a fair statement that Google is indeed on its way to being considered an ultra-large scale software system, and thus makes a perfect case study for the question of whether or not ultra-large systems still follow the design hints presented by Lampson almost 30 years ago.

# Lampson’s Hints

In his paper [8], Lampson outlines a number of hints for computer system design, and although these were provided almost thirty years ago, they still hold true in many current systems. While Lampson warns that the hints are not novel, foolproof, precise, or even always appropriate, his aim was to provide designers with a set of hints in order to design a good system. Through this section, I will present the three main categories (Functionality, Speed, and Fault-Tolerance) and the specific hints in each that are demonstrated by Google’s design. It is my hope that this will indeed prove that Lampson’s design hints still hold true on a system approaching the ultra-large designation.

## Functionality

The first area of hints provided by Lampson is functionality; the need for the system to do the things it is needed to do. Lampson’s first hint is to “keep it simple”, to which he adds to do one thing at a time and do it well, and to not generalize. This is an area where Google tends to flourish, by dividing their services into separate entities; Google is better able to work at the single task at hand. If someone wants to watch videos online, they can go to YouTube and watch videos, nothing more, just videos. The Google search engine is simple, it doesn’t do anything more than search on provided parameters. Each of Google’s individual services follow this hint, leading to the fact that Google as a whole certainly does “keep it simple”.

The second hint as part of functionality is to “make it fast, rather than general or powerful”. While this does have a lot to do with the second area, speed, it is also a focus of functionality; the removal of focus from generality and power leads to a faster system. Google again is able to prosper in this regard by keeping their searches quite fast. This will be explored further in the next section.

The final hint in terms of functionality that Lampson recommends that I will discuss in terms of Google is to “keep secrets”. It is important to not reveal everything about how your system works, such that it can be reproduced. Google does this really well on the development side, allowing those outside of the company to know only what is necessary to interface with the Google services and nothing more.

Google demonstrates a number of the hints outline by Lampson in terms of Functionality, showing that in that sense, the ultra-large systems of today still do follow the same design hints outlines almost thirty years ago.

## Speed

Google is fast; there is no doubt of that. One of the main reasons for the speed comes from the fact that Google follows one of the hints Lampson gives on speed, and that is to “cache answers” [9]. Google’s search engine does exactly that; pages are collected by web crawlers, and passed off to Google’s indexer which stores the pages in the alphabetically indexed database. From there Google’s query processor takes a query and searches the cached pages, and displays them to the user using their PageRank algorithm to rank the pages. (As an example, a search for "Hints for Computer System Design" was able to return accurate results in 0.35 seconds)

Another technique implemented in the Google search engine that aligns with Lampson’s hints is to “compute in the background when possible”. Google does this really well with their search engine by beginning queries as the user types into the search box. The user is still able to edit the query in the foreground while the search engine begins looking up pages in the database based on what has been entered. This background computation allows for faster results to the user.

Based on the following of these particular hints, it is clear that Google uses similar design processes for speed as other systems using Lampson’s hints.

## Fault-Tolerance

When discussing fault-tolerance, the focus will be on the Google File Server (GFS) [5], as this is where much of the design architecture for fault-tolerance is built into Google. Google’s two main areas for fault-tolerance (including fault avoidance) are to maintain a high availability and data integrity. One of the neat features used by Google to avoid data loss, and to ensure availability is their chunk replication, which ensures that data is stored on multiple chunkservers on different server racks, such that if one fails there are other copies readily available; the default number of copies is three [5].

One of the design principles suggested by Lampson in terms of fault-tolerance is to “log updates”, which is something Google does quite often as part of their replication of data, particularly their master replication, creating redundant copies of logs to ensure constant availability; these replicated logs are used to restart a new master process elsewhere immediately following a failure. This extensive use of logging and the added replication of logs allows for a high availability of the system.

Another of Lampson’s design hints is to “make actions atomic”; this is shown in the design of the GFS through their atomic record appends [5]. This is the default method of writing data, as opposed to a traditional write in which the client specifies the offset at which the data is written. In terms of fault tolerance, the atomic record append ensures that the data being written cannot be interrupted, and is only committed when fully complete, so as not to leave the file in an unstable state in the event of a failure during the write.

# Conclusion

Google as a system is designed to “keep it simple”, while still “mak[ing] it fast, rather than general or powerful”. Google is also good at “keeping secrets” by providing enough information to users and developers, but no more than needed. One of the main reasons for Google’s success is its speed in searches, which can be attributed to “cach[ing] answers” in their database to allow for an indexed search; this coupled with their ability to “compute in the background whenever possible” provides users with excellent speeds for internet searches. Lastly in terms of fault-tolerance, Google excels in their ability to “log updates”, but more so in their redundant storage of these logs, mitigating failure through multiple copies used to immediately restore functionality following a failure. In terms of the actions to the Google File Server, Google certainly took Lampson’s hint to “make actions atomic” by implementing an atomic record append that ensures a stable state of files even during a disk failure.

Given that the current modern day software giant Google, which could be considered a ULSS, follows these design hints presented by Lampson, our initial claim that the design paradigms intended for average or even large systems still hold for those classified as ultra-large; for now at least.

# References

1. Linda Northrop, “Ultra-Large-Scale Systems: The Software Challenge of the Future”, June 2006
2. Google Facts and Figures [infographic] - <http://royal.pingdom.com/2010/02/24/google-facts-and-figures-massive-infographic/>
3. Google Data Center Locations - <http://www.google.com/about/datacenters/locations/index.html>
4. Disaster Recovery by Google - <http://googleenterprise.blogspot.ca/2010/03/disaster-recovery-by-google.html>
5. Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung, “The Google File System”, 19th ACM Symposium on Operating Systems Principles (SOSP ’03), 2003
6. Luiz André Barroso, Jeffrey Dean, Urs Hölzle “WEBSEARCH FOR A PLANET: THE GOOGLECLUSTER ARCHITECTURE”, Microsoft IEE Volume 23, Issue 2 (22-28), 2003
7. Sergey Brin, Lawrence Page, “The anatomy of a large-scale hypertextual Web search engine”, Proceedings of the seventh international conference on World Wide Web 7 (WWW7) (107-117), 1998
8. Butler Lampson, “Hints for computer system design”, ACM Operating Systems Rev. Volume 15 Issue 5 (33-48), 1983
9. How Google Works - <http://www.googleguide.com/google_works.html>