Google Cluster Architecture overview

Paper link L2

Section C of the course web page

Disclaimer: *These abbreviated notes DO NOT substitute the textbook for this class. They should be used IN CONJUNCTION with the textbook and the material presented in class. If there is a discrepancy between these notes and the textbook, ALWAYS consider the textbook to be correct. Report such a discrepancy to the instructor so that he resolves it. These notes are only distributed to the students taking this class with A. Gerbessiotis in Fall 2016 ; distribution outside this group of students is NOT allowed.*

(c) Copyright A. Gerbessiotis.

We have already examined the state of a search engine (Google@1998). We present below some data taken from Subject 3. Roughly the size of the corpus (document collection) compressed was the size of the inverted index (barrels only) excluding links, anchors etc which can increase the total index-related data structures by roughly 50%.

A1. State of Google @ 1998 (research project at Stanford.

- Indexing of roughly 25M = 25,000,000 URLs.
- Average document size (text content) is roughly 8KiB or page size (everything) roughly 20 KiB (The "everything" includes HTML plus images, scripts, style sheets.)
- Total document size 147GiB uncompressed; 50GiB compressed ; 1 copy
- Index size (inverted index; barrels only) roughly 40GiB; All $\approx$ 60GiB ; 1 copy
- 1-2weeks to crawl; 1 day to index.

A2. State of Google @ 2003.

- Indexing of several billion URLs i.e. 1G-10G ($\times$100 increase over 1998).
- Average document size is roughly 25KiB; page size is roughly 130 KiB.
- Total document size 10sTiB uncompressed ; Dozens of copies
- Index size of 1s-10sTiB ; Dozens of copies
- 1month to crawl'update; 10 days to index

An estimate of the Surface Web around 2003 was in the order of 167 Terabytes, which is consistent with the "tens of Terabytes" cited in the 2003 paper.

(c) Copyright A. Gerbessiotis. All rights reserved.

A3. State of Google @ 2016.

- Indexing of 25G-100G of URLs ($\times$10 increase over 2003).
- Average document size is roughly 100KiB; page size is roughly 500KiB-1200KiB
- Total document size 10sPiB uncompressed; multiple copies.
- Index size of 1s-10sPiB; multiple copies
- Crawling frequently (Caffeine technology); Indexing takes few hours.

	1998	2003	2015
Docs	25M	3G	25G
PageSize(KiB)	20KiB	130KiB	1000KiB
DocSize(KiB)	7KiB	25KiB	100KiB
CorpusSize(Doc)	150GiB	10sTiB	1sPiB
IndexSize	60GiB	1sTiB	1sPiB
IndexUpdate	>1mon	1mon	>1min
IndexCycle	>1day	>10days	>1-2hrs
CrawlCycle	>7days	>10days	Demand/Prediction
Queries/day	1K	10M	4G
DRAM(typical)	256MiB	1GiB	64GiB
Hard-disk(HDD)	10GiB	100GiB	4TiB
CorpusSize(HDD)	16HDD	1000HDD	250-1000HDD
IndexSize(HDD)	6HDD	100HDD	250HDD

A4. Comparison through the years

(c) Copyright A. Gerbessiotis. All rights reserved.

A5. Parallel computing is the concurrent manipulation ("in-parallel") of data distributed on one or more processors to cooperatively solve a computationally intensive problem faster than other methods. A **parallel computer** is a multiple-processor computer capable of parallel computing. Sometimes the term parallel computer is confused with the term supercomputer. The term **supercomputer** refers to a computer that can solve computationally intensive problems faster than traditional computers. A supercomputer may or may not be a parallel computer. There are several forms of parallel computing.

- *instruction-level parallelism* (ILP), in which several operations are executed simultaneously employing techniques such as pipelining, superscalar execution, speculative execution, and branch prediction to exploit ILP,
- *data parallelism*, in which data are distributed over a number of processors.
- *task parallelism*, in which processes (or threads or a computational task) are distributed over a collection of processors.

A parallel computer can be a very complex highly specialized system or just a collection (usually referred to as a cluster) of PC workstations connected through a high speed (e.g. Ethernet-based) switch. It can be a **symmetric multi-processor** (SMP) in which several (identical) processors access a single (shared) memory, or a **Chip Multi-Processor** (CMP) that contains several control units (processors that are usually called cores) on a single die/chip. A CMP or multi-core processor differs from a **superscalar processor** in the sense that it maintains multiple instruction streams from which multiple instructions can be issued. In superscalar design, there is only one instruction stream. The cores within a CMP can in fact be superscalar processors themselves. **Multithreading** (e.g. Intel's Hyperthreading) is a form of virtual multi-core approximation, in which multiple threads are using one execution unit.

A **distributed computer** is a parallel computer in which the multiple (homogeneous or non-homogeneous) processors/-computers (i.e. processing units or elements) are connected through a high-speed external (to the processors, or cabinets housing the processors) network. In a cluster, the computers are loosely coupled and are more frequently non-homogeneous. **Beowulf clusters** refer to homogeneous, identical computer connected with a TCP/IP (fast) Ethernet network.

Specialized parallel computers come in the form of **reconfigurable computers** such as **Field-Programmable Gate Arrays** (FPGA) that act as a coprocessor to a general-purpose processor, **General-Purpose Graphic Processing Units** (GPGPU) that serve the same purpose that an FPGA serves, except that they specialize in graphics (vector-based) processing.

(c) Copyright A. Gerbessiotis.

Parallelism: Program vs Data

One example in real life that illustrates the principles related to parallel computing is book shelving in a library. If the number of books to be shelved is quite large, this task might take plenty of time to complete by one person. If one wants to speed up the process, one can take advantage of the inherent **parallelism** in this task and some form of **parallelization** is then used!

A6. Idea 1. A number p of workers is available. Then n books that are to be shelved are split evenly among these workers so that each one stacks about n/p books. If such an approach is used, several problems may occur when 2 or more workers attempt to shelve books in adjacent/consecutive locations. Some kind of arbitration is required on who works first or enters the stack first.

Question 1. Does it matter if some books are heavier than others?

Question 2. Does it matter if some books are located in adjacent locations?

A7. Idea 2. The p workers dealing with n books are assigned n/p books in such a way that no two workers work on the same shelf. This way, no book from two different workers will end up in the same shelf.

Question 3. How more complex is the splitting task? Does it require too much time?

A8. Conclusion. The most important issue in parallelizing a sequential task is identifying the inherent and apparent parallelism in that task, and then developing methods to exploiting it in an efficient manner. The following become then important aspects of the process of parallelization.

1. **A9.1 Task/Program Partitioning.** It describes the process of splitting a single task among the several available processors so that each processor is assigned approximately the same workload, and all processors can and will work collectively to complete the task. The object of task partitioning is that the processors finish the work faster than the time it would take a single processor to perform that same task. This is also referred to as **task parallelism**.

2. **A9.2 Data Partitioning**. It describes the potential splitting of the data used by the task among the available processors in such a way that processor interaction is minimized and the processors complete the task concurrently faster than an individual processor. This is also referred to as **data parallelism**.

3. **A9.3 Communication/Arbitration**. It describes the process of data exchange among the processors and how one can arbitrate communication-related conflicts.

(c) Copyright A. Gerbessiotis.

The need to satisfy queries fast requires,

B1. Multiple "Data Centers" geographically dispersed. Each center has one (or usually several) **clusters** of Distributed Storage servers. (Google for example maintains several such centers in SC, IA, GA, AL, OK, NC, TN, OR, and in countries such as Chile, TW, SI, Finland, Belgium, Netherlands and Ireland.)

B2. Multiple Copies of the ('Text'-based, 'Doc'-based) Web. Availability of a (or multiple copies) of the Web (text-based content, not graphics) close to where the query will be processed. Sevel copies of the Web per Data Center. Each copy is also replicated for redundancy/fault-tolerance and load-balancing and maintained by "storage-servers".

B3. Multiple Copies of the Index of the Web. Likewise, multiple copies of the index to satisfy multiple queries simultaneously. Each copy is also replicated for redundancy/fault-tolerance and load-balancing and maintained by "index servers".

B4. Task/Data Parallelism in query/index processing. Capability to answer a query within a subsecond response time (i.e. $\approx$ 0.25secs). requires the use of parallel computing in query processing that also involves index processing.

B5. Positive user experience (i.e. few delays, high throughput, high availability).

B6. Scalability of future demand (increase number of queries satisfied, increase data size seamlessly by increasing number of clusters).

In summary

- multiple copies of web
- multiple copies of index
- multiple clusters in multiple locations
- "parallelize" copies of web (data distribution)
- "parallelize" copies of index (data distribution)
- "parallelize" processing of copies of web (program or task distribution)
- "parallelize" processing of copies of index (program or task distribution)

(c) Copyright A. Gerbessiotis. All rights reserved.

B7. A simple calculation. Google deals with 4G queries per day, or roughly 50,000 queries per second. A query response time is roughly about 0.25secs. 80% of the queries are satisfied through caching of the results of already asked queries. Yet the are are still 12,500 queries that need to be answered at any give second. If 4 of them are satisfied per second, then thre need to be close to 3000 "copies" of the index to satisfy the remaining queries that need access to the index. By the last line of Table (Figure) A4, a copy of the index size requires at a minimum 250 Hard Disk Drive equivalent space. So collectively close to a million servers in Google are to be dealing with index related activity, if not more (to account for replication of data to increase availability and fault-tolerance multiply this estimate by a small number such as 3 to 4).

B8. ISSUE-1: Index distribution over multiple servers.
Let C be the corpus and let $I(C)$ be the index of the corpus. There two ways one can distribute $I(C)$ over hundredsof servers.

B8.1. Split-corpus and use Small In-server indexes independently. Split C into several (say k) pieces $C = C_1, \ldots, C_k$, where C_i, $i = 1, \ldots, k$, so that each piece C_i can fit into the HDD space of a single server (Say 25G documents are split into 1000 pieces of 25M documents each.) Then build $I(C_1), \ldots, I(C_k)$ i.e independently build the indexes of each subcorpus, so that $I(C)$ becomes $I(C) = I(C_1) \cup \ldots \cup I(C_k)$.

A problem with this approach is that a query such as `data AND structures` must be broadcast to all servers maintaining $I(C_1), \ldots, I(C_k)$, all servers need to process that same query and the results generated need to be transmitted by several (hundred or thousands of servers) to one or few "gatherer" servers that would "reorder" the results according to PageRank and further forward them.

B8.2. Split-index and use parallel processing for fragment processing. The alternative is to split $I(C)$ into multiple (say m) fragments $I_1(C), \ldots I_m(C)$, so that each fragment can fit into the HDDD space of a single server; index-size is a fraction of a doc's-size so m should be less than k as well. Then $I(C) = I_1(C) \cup \ldots \cup I_k(C)$. There is however a problem with this approach. Consider the index for the index-terms "data" and "structures" distributed over 2 servers for the former term and 4 servers for the latter. Trying to find the documents that contain both term becomes a complex "intersection" task that spans several not necessarily "adjacent" servers. The B8.1 approach does not have to deal with such complex processing issues!

```
  data     |  1 13 23  |  24 35 46 |                       Server 1 for data needs to intersect its list
structures |  1  2 12  |  13 14 15 | 18 19 23 | 24 35 44 |    with Servers 1,2,3 for structures ...!
```

B9. ISSUE-2: Web distribution How do we split the Web? How do we maintain a copy of the Web?

B10. ISSUE-3: Scalable design The Web doubles i.e. index also doubles in size. How does this affects the design choices? User demand quadruples. How does this affect the design choices?

B11. ISSUE-4: Optimization of Processing phases In B8.1 we mentioned that results transferred to a "gatherer" server might need to be reordered based on PageRank. (Traditionally, results are sorted by docID.) How can one avoid such a reordering? Google recently (2003) has been assigning `docID`s based on PageRank. A document with `docID` 1000 is higher-ranked (i.e more important) than a document with `docID` 1000000. Thus using such a "cheap optimization method" the reordering by PageRank can be avoided altogether. When documents are ranked by `docID` are also ranked implicitly by PageRank as well!

(c) Copyright A. Gerbessiotis. All rights reserved.

A solution to these problems is

(1) **C1. Geographically Dispersed Data Centers.** Disperse data-centers geographically to satisfy local demand, load-balance queries, minimize response time (latency to end user), and locate data-centers in such a way to also minimize electricity costs or disruptions (eg conflict).

(2) **C2. Inexpensive Clusters of PCs in a Data Center.** Connecting thousands of inexpensive PCs/servers interconnected with high-bandwidth network switches into a **cluster**. Organize and house multiple clusters in **data-centers**. A data-center can house tens of clusters of thousands PCs each. A Gigabit switch can accommodate up to 40 servers in a rack.

(3) **C3. Replication of the Web.** Store a copy (and replicate it for load-balancing and fault-tolerance) of the Web in each cluster.

(4) **C4. Replication of the (inverted) Index of the Web.** Likewise for the index; an index might be smaller so more copies might be used for the same space. Multiple copies for fault-tolerance, load-balancing, parallel-computing and fast response-time, high-availability and throughput.

(5) **C5. Redundancy fault-tolerance everywhere** at all levels of the design to achieve high availability, short response times, and high throughput. Though dispersed data-centers and multiple clusters satisfy this requirement, additional redundancy is required per cluster. Multiple copies of web and index within a cluster satisfy this to some degree (one HDD fails containing a fraction of the index yet the cluster can still be used for query processing). Additionally, every server has connections to multiple switches and switches to multiple other switches (a switch failure is manageable). Complex server and switch interconnection networks to avoid delays from faulty equipment, scheduled maintenance, etc.

(6) **C6. Parallel Computing.** Use several severs (eg B8) in the cluster cooperatively to answer queries (**index servers**) and to prepare results of queries (**docservers**).

(7) **C7. Maintenance and Electricity costs.** Keep equipment costs low, and maintenance costs also low (including cost of electricity) by strategically locating the data centers.

(c) Copyright A. Gerbessiotis. All rights reserved.

Types of servers employed by Google

A search engine uses multiple clusters, geographically dispersed to avoid availability problems caused by ordinary events (e.g. intense user activity) or extraordinary events (e.g power outages). The collection of servers of a cluster will cooperatively work to perform a single task (**parallel computing**) such as indexing, crawling, or query processing. A cluster in Google consists of several hundred/thousand PCs. Each cluster is capable of handling several hundred queries per second. For each query several of these PCs are being used to process it. Energy efficiency becomes important to maintain such clusters, and load-balancing techniques are employed at multiple levels (to be explained).

D1. Composition of Google Clusters. A Google cluster consists of a collection of PCs that are assigned distinct tasks. Among the PCs of the cluster we have

(1) **Google Web Servers** for processing user queries.

(2) **Ad-servers** that communicate directly with the GWS assigned to a specific query. (One ad-server per GWS.)

(3) **Spell-check servers** that communicate directly with the GWS assigned to a specific query. (One spell-check server per GWS.)

(4) **Cache servers** that process queries without accessing the index. (Multiple cache servers per GWS.)

(5) **Document servers** or **doc-servers** that communicate with a GWS in order to prepare the query response in HTML format with all relevant information about a link (URL rather than docID, title page information, context). Several document servers per query and GWS.)

(6) **Index servers** that process a query communicated by a GWS, and return an ordered collection of docIDs (doclist). (Several index servers per query and GWS.)

D1.1 Simplicity in replication over Complexity in hardware. Software reliability through straightforward replication and associated fault-tolerance is preferred over RAID-based solutions (Redundant Arrays of Inexpensive Disks).

D1.2 Simplicity in replication vs Expensive Hardware Complexity for reliability Replication is used not just for fault-tolerance but also to increase throughput and system availability. It is accepted that components will fail and such failures are anticipated and dealt with simply and cheaply.

D1.3 Sustained vs Peak Performance: Cheap up to a point vs Expensive. If the question is Price (cheap but slower machines) over Peak performance using specialized expensive hardware, the choice has always been Price. But there is a limit to this as well.

(c) Copyright A. Gerbessiotis. All rights reserved.

D2. Google Domain Name Servers. They are specialized machines that resolve domain name requests into IP addresses. For example a request to the Google Web-page through its domain name `www.google.com` will be first sent to a Domain Name Server which will then convert it into an IP-address. A DNS sever can resolve the domain name to the same IP address or to one from a small collection of IP addresses. This is used for example for load-balancing for a popular domain-name. This collection of IP addresses can also change from time to time. As of this writing, in Newark, NJ, the resolution of the domain name `www.google.com` is to one of the following 6 IP addresses: `74.125.131.{99,103,104,105,106,107}` . Let for the sake of an example, get it resolved to IP address `74.125.131.99`.

LB1 Load Balancing at DNS level. Google uses IP addresses of load-balancers of clusters to resolve the domain-name `www.google.com`. What a user types `www.google.com` the DNS acts as a **load-balancer** `[LB1]` and determines based on the geographical or network proximity of the user, the availability of one or more clusters and the load-balancer of one of those clusters is then forwarded to the user. In our example the user is directed to the cluster whose load-balancer is `74.125.131.99` and from that point on all user activity would be local to that cluster.

The load-balancer `74.125.131.99` monitors the local cluster and accepts queries from users. The queries are then forwarded to Google Web Servers (GWS). Thus the query `www.google.com/search?q=data+structures` resolved to `74.125.131.99/search?q=data+structures` will be forwarded to a GWS server among the many ones maintained by the cluster.

(c) Copyright A. Gerbessiotis. All rights reserved.

Query assignment: Google Web Servers

D3. Google Web Servers (GWS). There are several instances of Web-servers in a cluster that serve a single task: be the point of contact with a user's query. A Google Web Server is assigned to a particular query by the cluster load-balancer that is the contact point with the outside world, IP address `74.125.131.99`. This is a second `[LB2]` load-balancing mechanism performed not at DNS-level to pick up a cluster but at GWS-level to pick up a GWS among those GWSs available to a given cluster. The chosen GWS then interacts with
a **spell-check server**, a **cache server**, an **ad-server**, several **doc-servers**, and several **index-servers**.

LB2 Load Balancing at GWS level. The load-balancer contact point determines the server among the GWSs of the cluster with the lightest load that is available (i.e. not faulty, not in maintenance, not serving other queries) and assigns to it the user query.

In particular, the assigned GWS

- accepts a user's query and parses it,
- contacts a **spell-check server** to resolve typing errors,
- coordinates a query's execution with a **cache server** (more than 60% and up to 85% of queries are satisfied through a cache server) and determines whether a cached copy of the result(s) of the query already exists,
- if the query can not be satisfied by accessing a **cache server**, then **index servers** are used to satisfy the query and return results in the form of doclists i.e. sorted lists of **docID**s,
- and the GWS server further contacts **doc-servers** that is document servers that will facilitate in the formation of the output of the query in HTML format by converting a `docID` into a URL, obtaining the title page for the `docID` and the given URL, retrieve context text from the document file itself, and use it to construct the output HTML file, and finally generate the HTML query output that will be transmitted to the user, and if appropriate,
- contact **ad-servers** to generate ads specific to the query and the user, or maintain information about the specific session (user's IP address, habits, query, interests in specific results, etc).

(c) Copyright A. Gerbessiotis.

D4. Index Servers.
Phase 1 of query processing: Generate a doclist.

GWS uses index servers to process a query and generate a doclist which is a sorted sequence of `docIDs` that are relevant to the query. The order can be by "value" of the docID, or more frequently by the "relevance score" or "rank" of the document relative to the query. For every query term that is an index term, multiple index servers will access the index (i.e. inverted index) and retrieve the hitlists, decompress them if they have been compressed (Golomb-b, Elias-δ, or Elias-γ) and establish its doclist i.e. the list of docIDs that contain the index term. Then the index servers perform intersection of doclists or other set theoretic operations and compute a **relevance score** for each docID. The end result of the index server processing is an **ordered list of docIDs ordered by rank that includes not only the relevance score but also PageRank.**

Index servers can store the index or just process it. Most of the machines in a cluster are index servers.

LB3 Load balancing at index server level: Replication and Parallel Computing.
The same piece of the index is distributed potentially among multiple machines because it cannot fit into the memory of a single machine (or the disk assigned to a single machine). The same piece of the index is also replicated for redundancy and throughput and availability. A load-balancer (LB3) determines which one of the replicated pieces of the index will be used for the query. A piece of the index is called an **index shard**. It is the index of several randomly assigned docIDs. Pools of machines are assigned to each shard because it cannot fit into one machine, and one or more pools of machines is assigned to each shard for availability, scalability and throughput.

An index shard is

- a random collection of docIDs whose index is stored in multiple groups (pools) of machines for throughput and availability, and
- popular index-terms have multiple index shards.

D5. Doc-Servers (document servers).
Phase 2 of query processing: Sending the doclist back to the user in beautified HTML format.

The sorted list of docIDs generated by the index servers is then converted into HTML format. Each docID gets converted into a URL. Title information, as well as the context of the document (i.e. the text around the index terms) that is relevant to the query and is query specific are then generated in HTML.

Document shards. The docIDs and the associated documents themselves, not the index of the documents, form a `document shard` that is maintained by the doc-servers.

Multiple copies of a given document shard. The doc servers contain a low-latency copy or multiple copies of the Web in the doc-servers. A load-balancer LB4 determines which copy will be accessed by a given query. More copies of popular or important documents might be maintained than lesser important documents.

LB4 Load balancing at doc-server level: Replication and Parallel Computing.

Important documents have more shards, important documents have small docIDs. The doc-servers maintain a low network latency copy of the web; in fact multiple copies might be stored in the doc-servers. In principle it makes sense to replicate more popular or frequently accessed documents than infrequently accessed ones and this is implemented by the doc-servers.

A document shard is

- a random collection of docIDs stored (documents themselves) in multiple groups (pools) of machines for throughput and availability,
- and popular documents have multiple document shards and also small docIDs.

(c) Copyright A. Gerbessiotis.

D6. Similarities/Differences between index and doc servers.

D6.1. Most numerous types of servers in a cluster. Most of the servers of a cluster are index and document servers. Index servers and document servers have the same fundamental architecture i.e. they are inexpensive PCs and there are multiples of them for throughput and also availability. However there are some differences which might affect the type of PCs (if a non-homogeneous collection is maintained) that are assigned to become an index server or a doc-server.

D6.2. Doc-servers for I/O. For doc-servers disk-space is important but speed is not as important. They are read-only, occasionally-write devices.

D6.3. Index servers for CPU performance. For index servers speed is more important than (hard-disk) space. The most time consuming task an index server performs is decompressing occurrence lists/hits, i.e. they are read-write devices.

D6.4. Satisfying one more query. The incremental cost of an additional query requires a heavier investment on index servers rather than doc servers. The index needs to be replicated once (and for fault-tolerance) or more and more index servers need to be used to process or generate the doclists from the compressed hitlists.

D6.5. Corpus Size vs Index size. In 1997 the size of the index was 40% of the size of corpus(approximately 60GiB vs 150GiB). Given that Google is using consistently **replication** to achieve redundancy and effect **high throughput, availability, fault-tolerance, and potential scalability**, a cluster of servers will maintain at least 2-3 copies of the Web.

D6.6. Pervasive use of parallelism. Parallelism is also used in a multiplicity of ways. Dozens of clusters are scattered around the world (LB1). Thousands of machines per cluster and GWS (LB2) and also several or dozens of copies of the web or index in the index and doc servers (LB3).

Note that document and index servers might or might not store themselves the document and/or the index. There are dozens of copies of the Web in the clusters (doc servers) and also dozens of copies of the index in the clusters (in index servers or machines attached to the index servers). Yet additional index servers might be used just for dealing with doclists.

(c) Copyright A. Gerbessiotis.

Query Processing: Scalability and Growth

Growth requirements are satisfied by

- adding new data-centers,
- adding clusters,
- adding new pools to existing clusters (i.e. adding machines to clusters),
- adding machines to pools,
- adding copies of existing (document or index) shards,
- adding a new shard for a new set of docIDs, and
- **splitting a shard**.

(c) Copyright A. Gerbessiotis.

Parallelism in Google

In Google, the search engine achieves parallelism by using multiple levels of task and data partitioning in a rather straightforward manner which is a direct implication of the discussion of the previous page.

(1) The creation of independent, geographically separate clusters is one form of task parallelism. Thus if there are four clusters and four queries are fielded, and all other proximity requirements are satisfied, one query will be fielded per cluster. In order to achieve this task parallelism, Google uses replication of data. Unless some synchronization mechanism is used, it is potentially possible, that the state of the Web in one cluster will be different from that of another cluster. Thus **synchronization** is important or the **lack of synchronization** may result to **consistency problems**, in which the same query on different clusters might yield different results or a different ranking of the same results.

(2) Task parallelism within a cluster is achieved by having multiple servers for indexing (the index servers) and also document processing (the document servers). Thus for example, a collection of index servers work together to satisfy a query. Otherwise, one server would have been impossible to satisfy the query within a given limited time-frame (e.g. subsecond response time). The same paradigm applies to docservers as well.

(3) Data partitioning is imperative. Docservers not only process doc-related requests but are affiliated with the storage locations that hold the copy (or copies) of the Web maintained by the cluster. Data partitioning is imperative, since a single computer or a single computer's disks memory. Therefore without data partitioning, it would have been impossible to implement a search engine. At the same time data partitioning facilitates fast response times. Given that multiple servers process different segments of the index or the Web, they can complete their independent tasks predictably quickly. The only bottleneck might be the combining final step when partial results are combined to form the end result.

(4) Communication/Arbitration sometimes requires a single contact point that serves as the **master**. For example a Google Web Server is the master and communicates with its subordinates ("slaves") that can be the ad-server, or the spell-check server, or the collection of index and doc servers. The GWS is the master because it is also the contact-point to the end-user.

(c) Copyright A. Gerbessiotis.

Parallelism in Google

However certain data partitioning methods in Google are used not only to achieve parallelism but also high throughput and high availability and also fault-tolerance. Towards this Google also does the following.

(1) The collection of `docID`s that are indexed or stored in the "Repository" and processed by the docservers are randomly split into smaller pieces. This is a form of data partitioning. The pieces become independent of each other and in Google terminology are called the **shards**. Thus an index is split into multiple `index shards`, that are indexed (up to a point) independently of each other (a common dictionary might be used for all the shards). Similarly the docservers deal with **document shards** as well. This is a form of data partitioning to facilitate task parallelism.

(2) A given `index shard` (i.e. index of a given collection of `docID`s) is stored multiple times. A collection of one or more machines is assigned to each shard. In addition one or more collection can be assigned to each shard. All the PCs assigned to a shard are known as the **pool** for the shard. Thus for example 1000 machines are assigned to the pool of a given shard, so that we have 10 collections of 100 machines. For query processing of the particular shard one of those collections is chosen (that uses 100 machines concurrently). This replication allows for higher availability. For example, if one PC of the collection goes down, it can be replaced by some other spare PC assigned to the collection or the pool. The same applies to the `document shards`.

(3) The splitting into `document shards` or `index shards` allows the scalability of the search engine architecture. When new URLs are presented to the indexer, a new shard can be created independently of the existing ones to serve the new URLs. A shard that grows large (say because the original URLs have grown in size) can be split into two or more shards. `Shards` can also be deleted if several docIDs become stale or disappear. If the cluster needs to grow for scalability to satisfy more user requests, then the `pools` can be increased by adding additional PC collections. If a pool consists of slower PCs you can increase its size to compensate for that.

(4) Because of the existence of shards, the servers can work on independent shards and thus little communication or synchronization is needed. Thus in this sense data parallelism (shards) can facilitate task parallelism as well since the task parallelism of the servers requires little or no synchronization or communication (among the servers).

Note. It is highlighted that the Google cluster architecture is designed so that it can perform quickly and efficiently mostly read-only related processing. Query processing does not modify the index or the document collection. Such modification is done periodically and separately.

(c) Copyright A. Gerbessiotis.

E1. Cost of Data Center Per MW. A Data-Center with 1MW installed power capacity costs approximately $15-25Million, and this cost scales accordingly. Electricity, air-conditioning, heating (dissipation) and cooling, and water costs are part of maintenance costs. The cost of electricity per kWh is roughly 6-10cents; one can use as an estimate a cost of 15cents per kWh consumed that includes all maintenances costs. Power losses are 67% due to servers, 11% to power transmission reasons, and 22% to water cooling, on the average. (Assumes a PUE of 1.5, to be explained in E6 and E7.)

E1.1 Data Center (DC) size and Wattage. Data centers of size 50,000-80,000 are common; very rarely does one see DC with much more than 100,000 servers. A DC nominal power is at 8MW or 15MW and rarely (see E5 below) at 65MW or higher. A rack typically has 20 servers. Racks with 10 hard disk drive enclosures of 800-900 HDD (Hard-disk drives) are available. 10kW-15kW are not uncommon rack power capacities. A network switch consumes 100-150W per port so 24-port, 48-port or 96-port switches are in the 3kW-15kW range.

E1.2 DC Networking. Data centers of size 50,000-80,000 are common; Network traffic incurs latencies 1-2ms within it. East-Coast to West-Coast latency can be 70-80ms. Software latency for the API of AWS (Amazon Web Services) is around 3ms as well. Typical network capacities (2014) are at 100Tbps per Data Center (or 1-10Gbps per server).

E1.3 Water cooling. For every 1kWh consumed, 0.5-1lt of water is used for cooling (1lt $\approx$ 0.25 gallons). Or 1MW installed power uses roughly 25,000 gallons per day.

E1.4 Hardware Failures. Typical annual failure rates for hard disk drives are 4-8% and for servers at around 4%.

E2. Acreage of a data center. In a data center only 10% to 30% of available space is used as rack-space with the remaining one occupied by auxiliary equipment, heat exchangers, air-conditioning, auxiliary power supplies, etc. A 1,000,000sqf data-center can accommodate approximately 3000-5000 racks of approximately 20-40 servers each or a total of 60,000-200,000 servers. Power consumption per rack can range from 10KW to 15KW. How many servers can be accommodated depends on the per server power consumption! A small server is rated for 200-250W with more than 400W for high-end servers or servers with multiple CPUs, hard-disk drives etc. Roughly 10% of the square area of a data center may be occupied by servers.

(c) Copyright A. Gerbessiotis. All rights reserved.

E3. Server Power Consumption and Utilization. Roughly a 1/4 of the consumption is by the CPU, 1/4 by the memory components (DRAM) and 1/4 by disk-drives, networking equipment, 1/4 are power supply and fan losses. (Servers with several disk-drives might have a slightly different power behavior.) On the average machines are at 10-20% utilization, and while idle at 10% utilization. Yet power consumption while a machine is idle is 50-60% of peak power consumption, and average power consumption is also 65-70% of peak power consumption. A typical 2015PC (16GiB RAM, 4x1TiB disks) requires at least 300W. A 300W power supply at 75% efficiency makes available 225W of power. A typical CPU consumes 60-90W (quadcore), with 4-8W per memory unit (1GiB or 4GiB) and 10-20W per disk or network card. A network switch has consumption of 150W per network port. So expect 7kW or 15kW per 48-port or 96-port switch.

E4. Standby Power consumption. Even "green" Data Centers need standby diesel generators that take over in case of power outages. Standby power generation is equal to peak power needs.

E4.1 Power Transmission and Distribution. At generating stations power is transmitted through 110kV lines. Transformers (10kVa) at 99.7% or so efficiencies get it down to 13kV. UPS units at 94% efficiency or so can provide stable 13kV output. Or midrange transformers (3kVa) at 98% efficiency bring 13kV down to 480V. Further transformers (98% efficiency) can bring 480V down to 208V. Standby generators (2.5KVa) usually provide 480V. Transmission losses are thus around 11% .

E5. Example. NSA (National Security Agency) built its Utah Data Center at a cost of $1.5billion, supported by a 65-MW electrical substation (Wired Magazine, 2012/3/15, James Bamford). About 100,000sqf of the 1,000,000sqf is server space according to
`www.forbes.com/sites/kashmirhill/2013/07/24/blueprints-of-nsa-data-center-in-utah-suggest-its-storage-capacity-is-less-impressive-than-thought/`. The NSA Bluffdale, Utah Data Center, in July 2013 had a consumption of 6.2 million gallons; it is thought that in full utilization it would consume 1.7million gallons per day! (See also E1 for water consumption.)

(c) Copyright A. Gerbessiotis.

E6. PUE for Power Usage Effectiveness (Typical values 1.3-1.6)

PUE for Power Usage Effectiveness is defined as the ratio of **total facility power** to the **IT equipment power**. The former includes the latter plus power consumption used for lighting, heating, and air-conditioning and other auxiliary activities. Roughly of the Total Facility Power, 30-50% is used by the IT Equipment, 30% by the chilling infrastructure, 10% by the CRAC (see below), and up to 20% by the UPS, lighting etc. The ideal PUE ratio is **1.0**. Any values **less than 1.6** is considered a good PUE value. In general Data Centers achieve a PUE close to 2-2.5 but Google claims a PUE ranging from a low of 1.16 to 1.25. As of this writing the average of traditional data centers is getting closer to around 1.6-1.9.

$$PUE = \frac{\text{Total Facility Power}}{\text{IT Equipment Power}} = \frac{\text{TFP}}{\text{IEP}}$$

E7. DCIE or DCE for Data Center Infrastructure Efficiency DCIE is defined as $1/PUE \times 100\%$. Thus if PUE is 1.5 then DCIS is 66% .

E8.1 Comment: Manipulation or Adjustment of PUE ratios. In order to minimize PUE one has to minimize the numerator or maximize the denominator. Thus for **TFP / IEP** to become smaller one could potentially subtract a term from the numerator and add it to the denominator so that the new PUE becomes **(TFP-a)/(IEP+a)**. Some companies use integrated containers that contain the PC hardware (IT equipment) but also integrated to it the components for air-conditioning and to provide lighting in the area of the container. Thus a portion a of the power for the facility is integrated into IT equipment power.

E8.2 Comment: More Terminology. Other terms used in the context of data center technology are **CRAH** for Computer Room Air Handler, and **CRAC** for Computer Room Air Conditioner.

SPUE (Server Power Usage Effectiveness) and TPUE

E9. SPUE for Server Power Usage Efficiency (Typical Values 1.6-1.8). SPUE for Server Power Usage Efficiency deals only with the consumption inside a server. It is defined as the ratio of total server input power over useful server power i.e. power delivered to computation units including disks, CPUs, memory (DRAM), motherboards and I/O units. The losses incurred are due to power supplies, fans, voltage regulators, cooling fans etc. An SPUE of 1.6-1.8 is common; a minimum for SPUE is 1.2.

$$SPUE = \frac{\text{Total Server Input Power}}{\text{Useful Server Power}}$$

E10. TPUE for Total Power Usage Efficiency (Typical values 2.0-3.0). The products of `SPUE` and `PUE` is known as `Total PUE` or `TPUE`. A ratio of 1.2-1.25 is probably the best possible and Google claims a 1.3-1.4.

E11. DCPE for Data Center Performance Efficiency. Finally in order to compute the **Efficiency** of the installation one need to determine the amount of computation performed for the energy consumed. The first term is related to the building efficiency and the second term to IT equipment efficiency.

$$\text{DCP Efficiency} = \frac{\text{Computation}}{\text{Total Energy}} = \frac{1}{\text{PUE}} \times \frac{1}{\text{SPUE}} \times \frac{\text{Computation}}{\text{Total Energy to Electronics Components}}$$

E12. Equipment utilization. Note that the average utilization of equipment in a data center varies but not by much from 10%-20%. Google claims a utilization of 10%-50%. A 30% utilization is considered very good!

(c) Copyright A. Gerbessiotis.

F1. Server Racks: General Information. In building a cluster, a collection of physical racks is being used. One 19inch rack is approximately 24in x 30in i.e. 5 square feet. A rack can accommodate components that are 19 inches wide and is a standardized frame for mounting computer modules. Fire regulations require a 3 foot hallway around them so 20 to 40 square feet is a good estimate of the amount of space occupied by a rack in a data center. Thus a 100,000sqf of rackspace in a data center can accommodate 2,500 to 5000 racks. The capability of a side of a rack is expressed in terms of **U**s. A module can be 1U or 2U high or multiples of 1U. 1U is 1.75 inches (a pizza-box type of machine). So a 40U side can accommodate 40 1U machines, or 20 2U machines. The lifetime of a rack (or its PCs) is roughly 2-3 years; beyond this either failures or older technology make the components of a rack unusable or uncompetitive. Static weight capacity is roughly 3000lb.

F2. Power load. Typically consider a load of 5KW/rack (20sqf) to a maximum of 15kW (40sqf). The Power density of a rack is anywhere between 200W/sqf to 750W/sqf.

$$15\text{KW/rack} \times 24\text{hrs/day} \times 30\text{days/month} \times \$0.15/kWh = \$1000\text{/month/rack}$$

(The electricity cost of 15cents per kWh includes the cost of electricity to power the rack, the cost of electricity for the air-conditioning, and the cost of the Uninterruptible Power Supply or UPS used for uninterrupted operation.)

F2.1 Rack Power consumption and reliability. A rack with 20 servers can have nominal consumption 250-400W per server i.e. 5-8kW. A Top-of-Rack 20-port switch is rated at 2-3kW. So the total is at the 8-15kW range. A rack with HDD enclosures supporting 800-900 HDDs is also in the 8-15kW range. A End-of-Row switch or a Core-switch with 48-ports or 96-ports (100-150W per port) is also in the 8-15kW range. On the average a rack experiences 1 server and 1-2 HDD failures annually.

F2.2 HDD sequential-write vs random-write performance. Be reminded that a 4TiB on serial-write mode one can write down a full drive's data in roughly 8-12 hours. Random-writes usually require 40-50 days for the same task. Thus sequential to random performance is 120:1. 10 years ago (2003) it used to by close to 80 to 1, in 1998 it was 20 to 1, and 1994 closer to 4 to 1.

(c) Copyright A. Gerbessiotis. All rights reserved.

F3. Networking equipment (ToR, EoR, aggregate, core switches). The Top of the Rack (ToR) is occupied by a switch which is thus known as the ToR switch (this ToR has nothing to do with TOR: The Onion Router). ToR switches are connected to EoR (End of Row) switches; the next level is aggregate switches, and then core switches.

F3.1. 2003 and 2015Network Switches. In 2003, the PCs on each side of a rack were connected to a 100Megabit/port full-duplex high speed switch (24-48 ports). Nowadays we have 24-48port 1-10Gbps switches. In 2014-2015, network switches can accommodate 20 PCs of a rack (or one side of) through 10Gbit switches. The two switches of a rack can then connect to a 10-40gigabit core switch that connects several racks together. Each switch is connected to the core switch with one or two 10-40gigabit uplinks. The uplinks are limited. Even nowadays, uplink capacity is limited.

F3.2. Oversubscription. The limitation on uplink capacity gives rise to the term **oversubscription** that describes the ratio

$$\text{Oversubscription} = \frac{\text{In network switch throughput}}{\text{Out network switch throughput}}$$

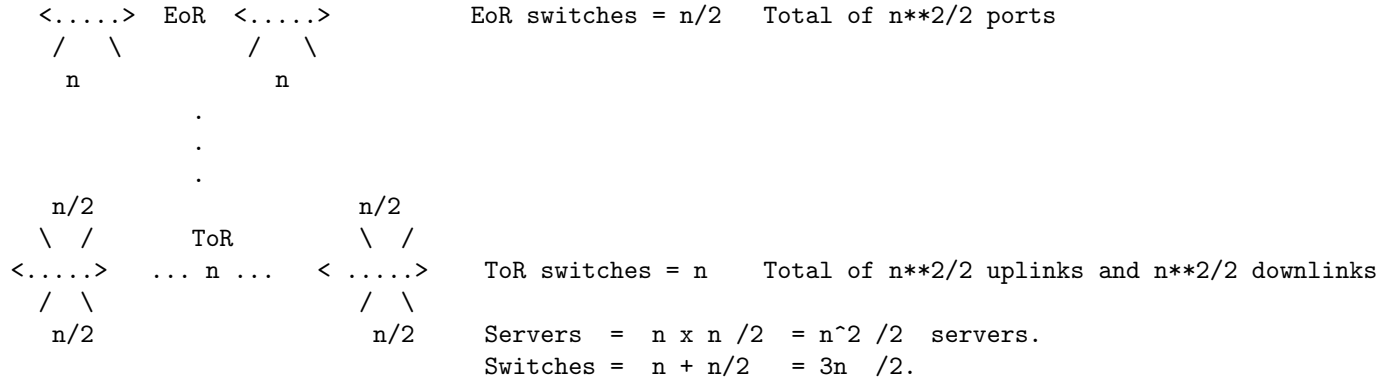
i.e. it is the ratio of the network throughput within the switch to the throughput of the uplinks.

F3.3. 2003 Google paper. So keeping things up with the paper a switch with 48, 100Mbps ports has throughput equal to $48\textbf{port} \times 100\textbf{Mbps/link} \times 2\textbf{links/port} \approx 9.6\textbf{Gbps}$. Yet 1-2 uplinks of 1Gbps might only be available. This translates to an oversubscription of $9.6/2 = 4.8$. An oversubscription of 5 is a often quoted number.

Network design:Simple Configuration

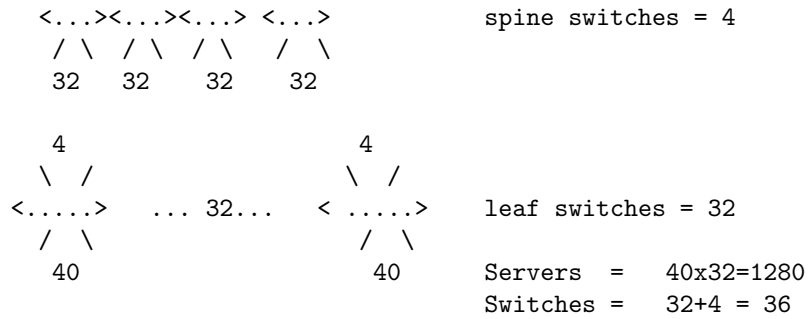
We show how one can design a small network with up to hundreds of servers in a one-level configuration that indcluded ToR and EoR switches. Let n be the port-number of a switch, and let us use homogeneous (same type switches). Each ToR Switch is connected to $n/2$ servers and has $n/2$ uplinks to EoR switches. The identical EoR switches use all of their n port as downlinks to the ToR switches. Each port of a ToR switch gets connected to a different EoR switch and thus we have complete connectivity. A total of $3n/2$ switches (EoR and Tor) and a total of $n^2/2$ servers are supported! The Oversubscription of the configuration is equal to 1.

```
                          n=24 : #switches 36 , # servers 288
                          n=48 : #switches 72 , # servers 1152
```



Design choices and implications: Modern design

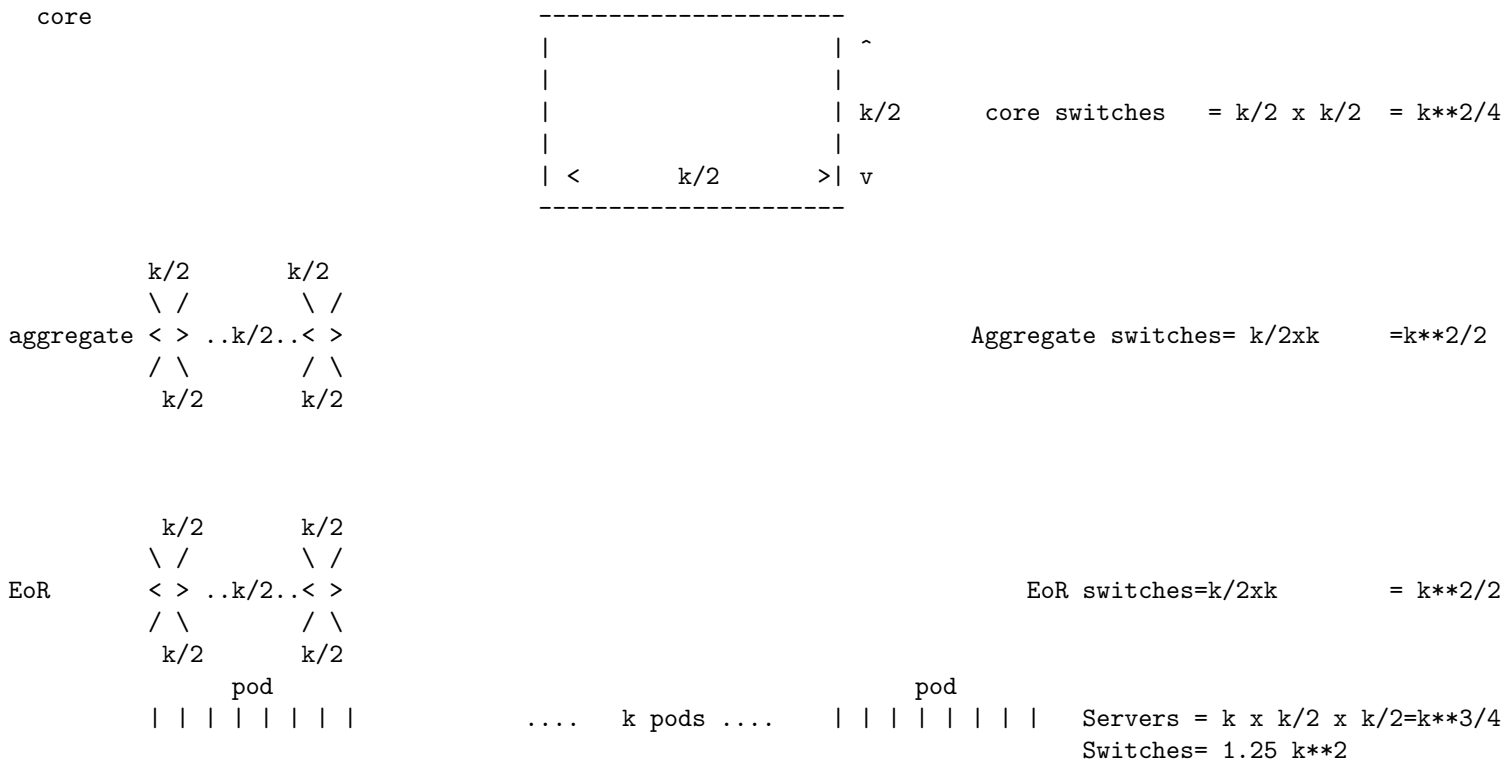
In this design we use 44-port switches. 40 of the ports are 10Gbps and 4 are 40Gbps ports that are used for uplinking. The oversubscription is 2.5 (as $40 \cdot 10/(4 \cdot 40)$). 32 switches are connected to 40 servers each. This is the "leaf" level. The 4 uplinks of a switch are connected to 4 different switches at the "spine" level.



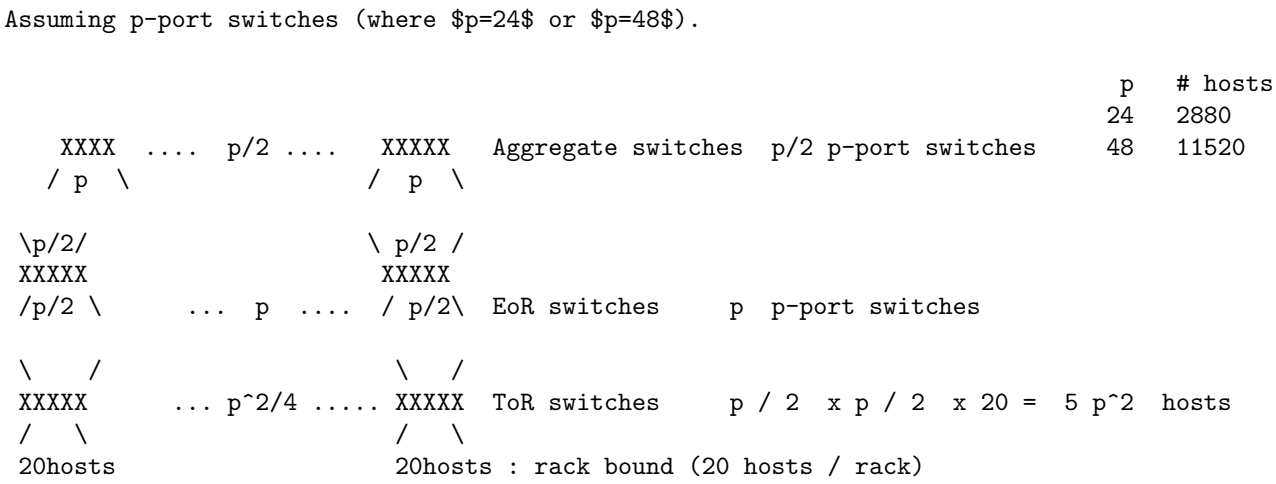
(c) Copyright A. Gerbessiotis. All rights reserved.

Design choices and implications: Pod design

A pod is a $k/2 \times k/2$ configuration of servers into $k/2$ rows and $k/2$ columns. There is an aggregation of k pods into a total of $k^3/4$ servers. For $k = 24$ this provides for 3456 servers; for $k = 48$ up to 27648 servers can be configured. Respectively, 720 and 2880 switches are being ussed. The columns (servers) of a pod are connected to a switch (EoR) with k ports utilizing half of the ports for that. The EoR switches are then connected to the same number of aggregate switches which in turn collectively are connected to a $k/2 \times k/2$ grid of core switches. Each EoR is connected to every one of the $k/2$ aggregate switches on "top" of it. The Uplink ports of a collection of $k/2$ aggregate switches is connected to each one of the core switches.



Design choices and implications: Network design



Design choices and implications: Network design

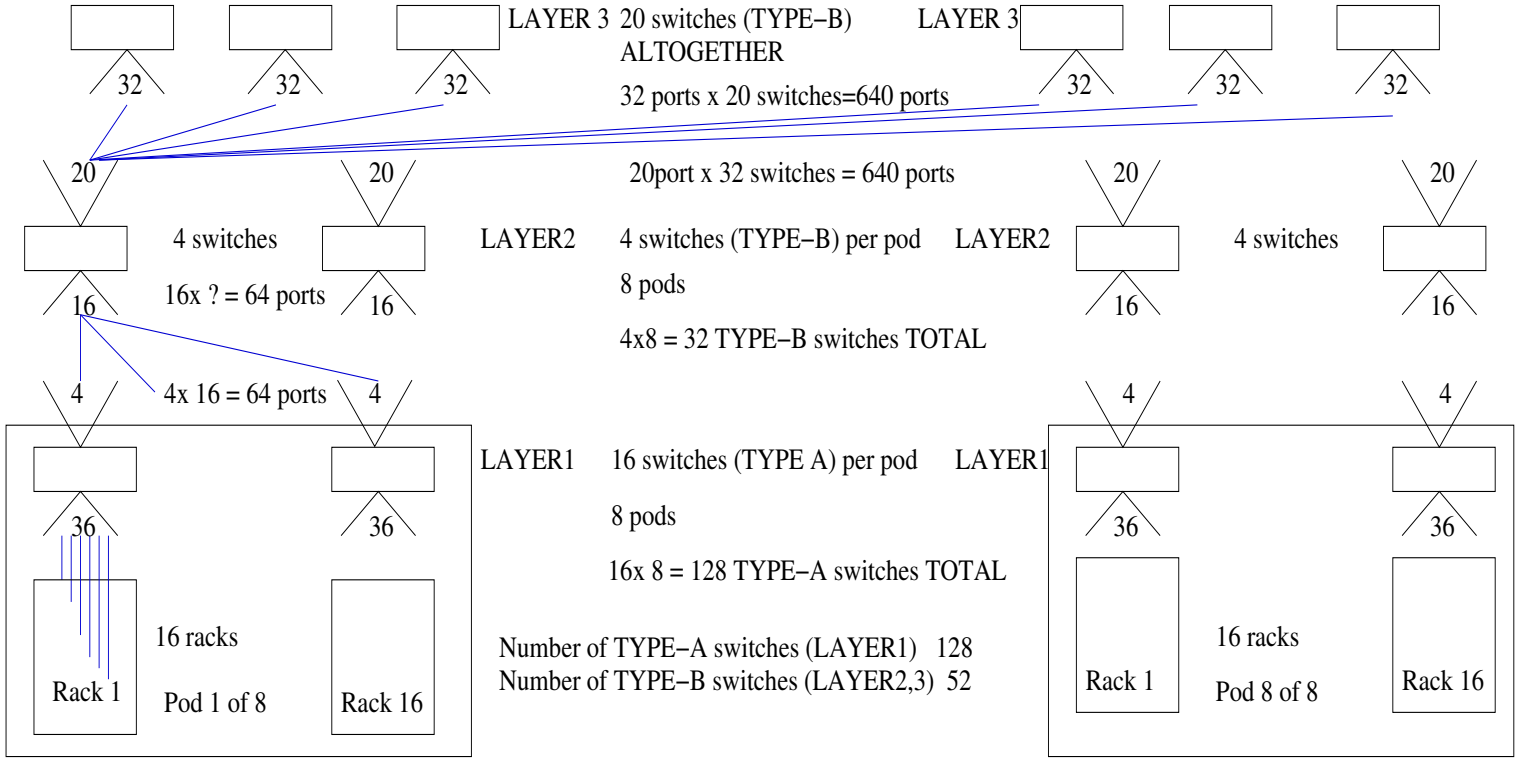


Figure 1: Facebook-type POD Design :2 levels , 3 layers (TOR, Aggregate, Core switches)

(c) Copyright A. Gerbessiotis. All rights reserved.

Exercise 1 *Read the paper on the* **Google Cluster Architecture** *by Barroso, Dean, and Holzle.*

Try to answer after reading the paper the following questions whose answers are in the paper or may need to be further thought upon. You are given 2000 machines (PCs). Using the data available in the paper answer the following questions in a way that are fully compatible with the paper. Read all the questions first since some answers (eg. for (c)) might depend on the answers/solutions that you suggest to some other questions (eg. for (a)). **(a)** *How would you organize the 2000 machines? How many racks are you going to use? How many network switches? Other resources? Explain.* **(b)** *What is the collective memory (RAM), disk space (Gbytes) of these resources?* **(c)** *Using data from other papers you have already read for this course, how many of these machines will be file servers, how many index servers, and how many query machines? How many queries can the structure support and how much information can it index (in number of documents, and Gbytes).*

Figure 2: Be careful with Exercise 2 :-)

Exercise 2 *NSA (National Security Agency) built its Utah Data Center for 1.5billion. It is supported by a 65-MW electrical substation (Wired, 2012/3/15, James Bamford). Estimate the server size of that facility and amount of information that can be maintained. Make sure your answer is kept brief (no more than a paragraph). Use 2014 data for typical PC configurations.*

(c) Copyright A. Gerbessiotis. All rights reserved.

**All images/material are taken from the source indicated in Figure 2 which can be obtained from https://wangxliang.wikispaces.com/file/view/Data-center-network-google.pdf
Some pictures (figures 4,5,6,7,8) were obtained from
http://webodysseum.com/technologyscience/visit-the-googles-data-centers/**

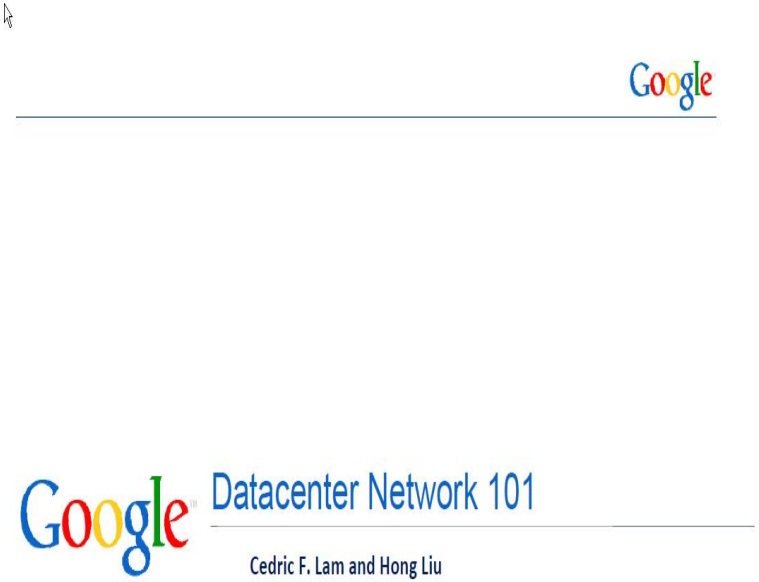


Figure 3: Source Reference

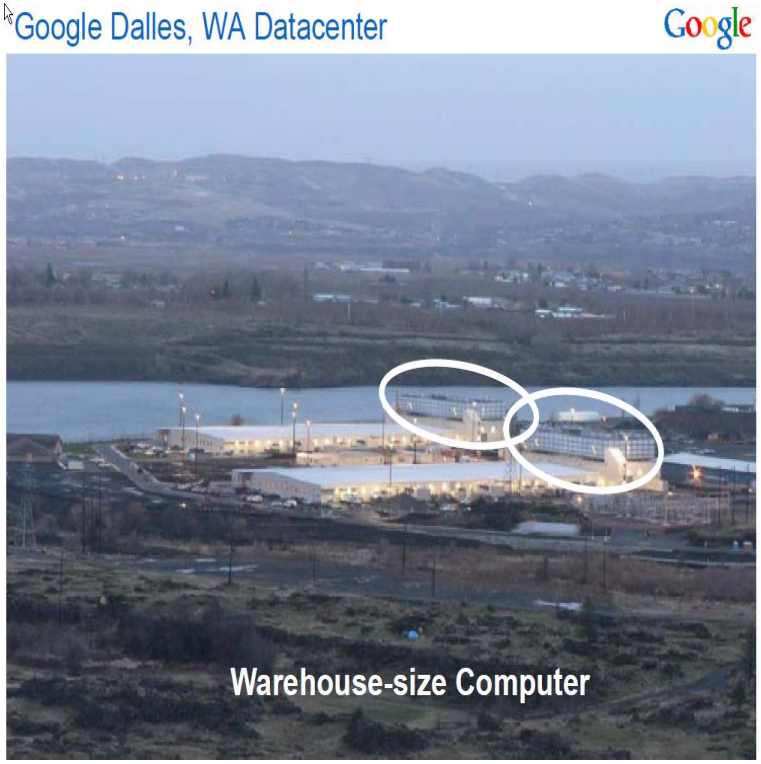


Figure 4: A Google Datacenter

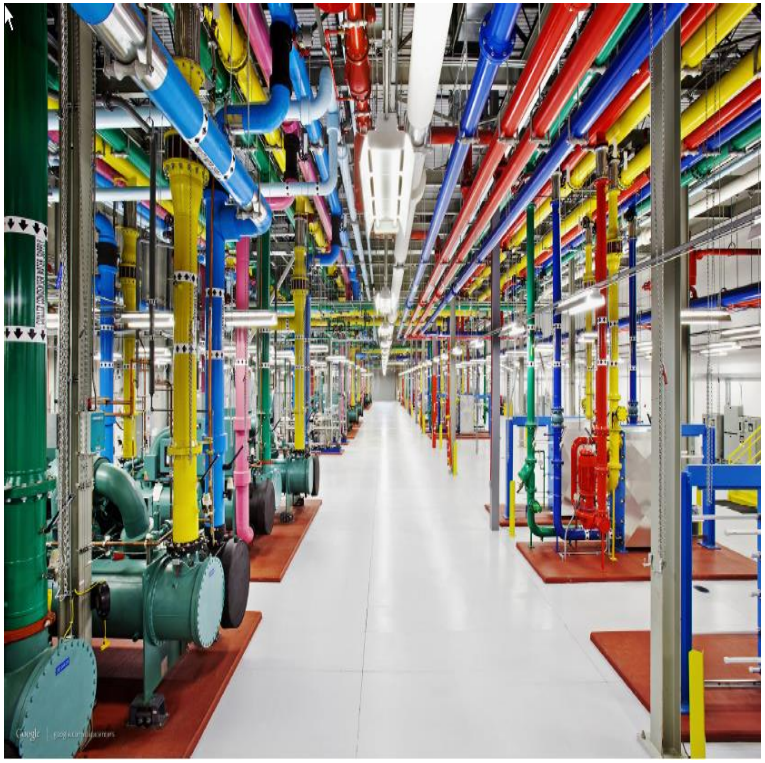


Figure 5: Cooling

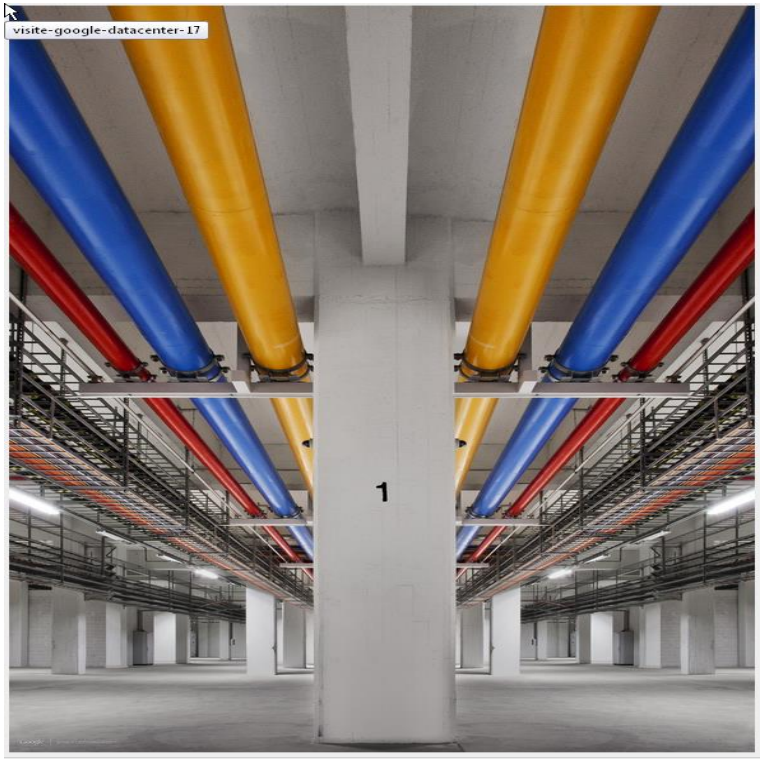


Figure 6: More Cooling

(c) Copyright A. Gerbessiotis. All rights reserved.

Figure 7: Cool stuff

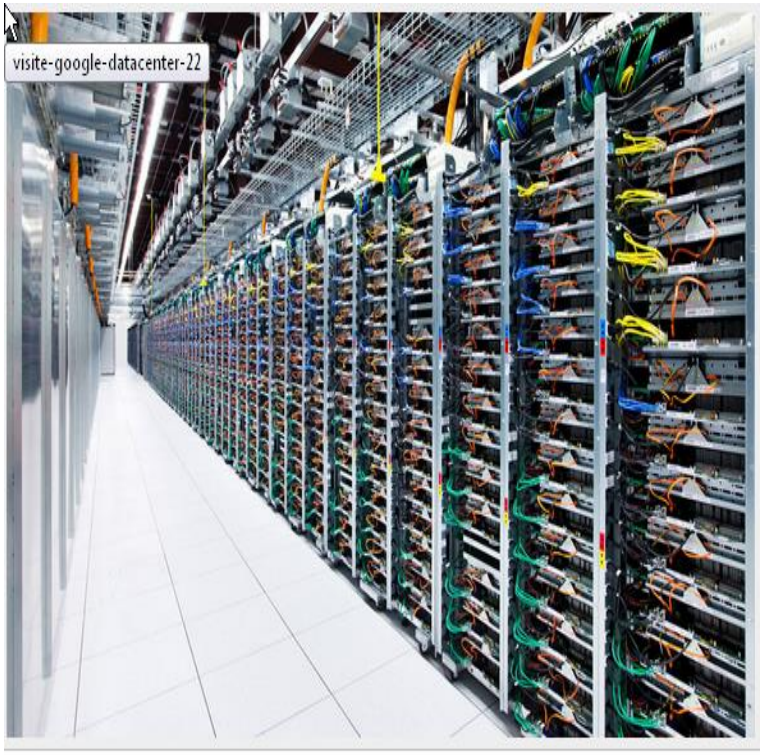


Figure 8: Equipment: Orderly structure

(c) Copyright A. Gerbessiotis. All rights reserved.

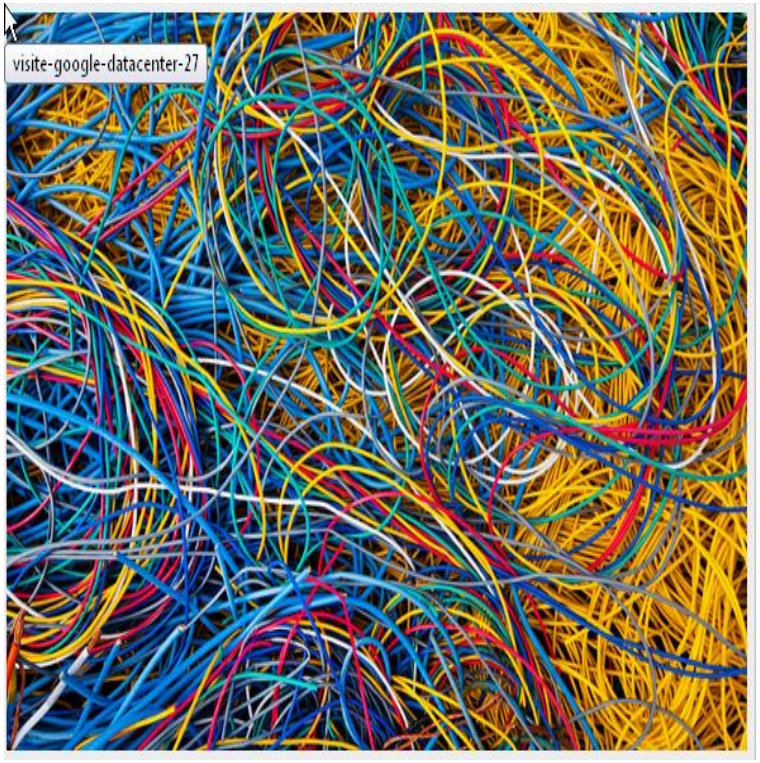


Figure 9: Find the yellow connection here

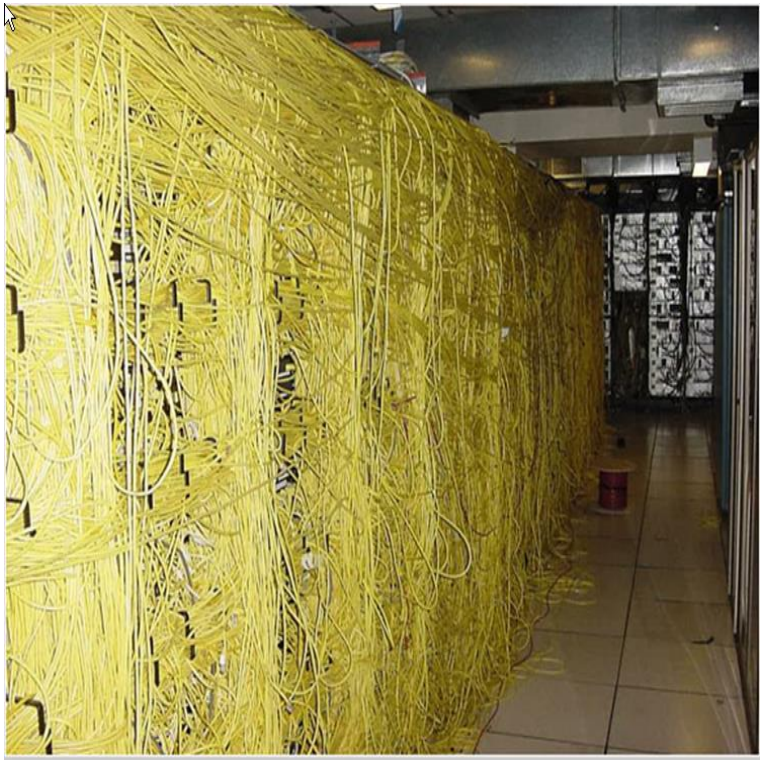


Figure 10: or there

(c) Copyright A. Gerbessiotis. CS345 : Fall 2016 . All rights reserved.

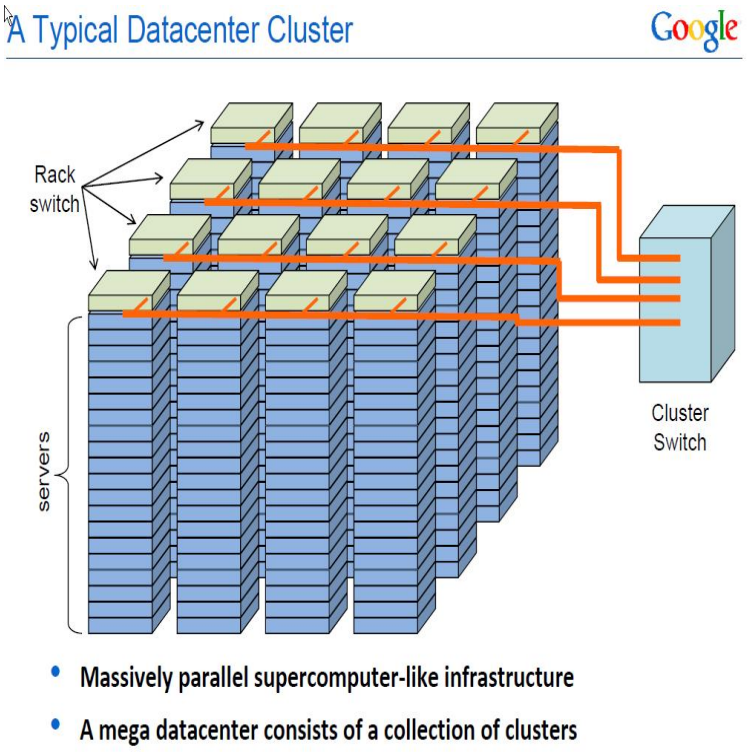


Figure 11: Cluster geometry/topology

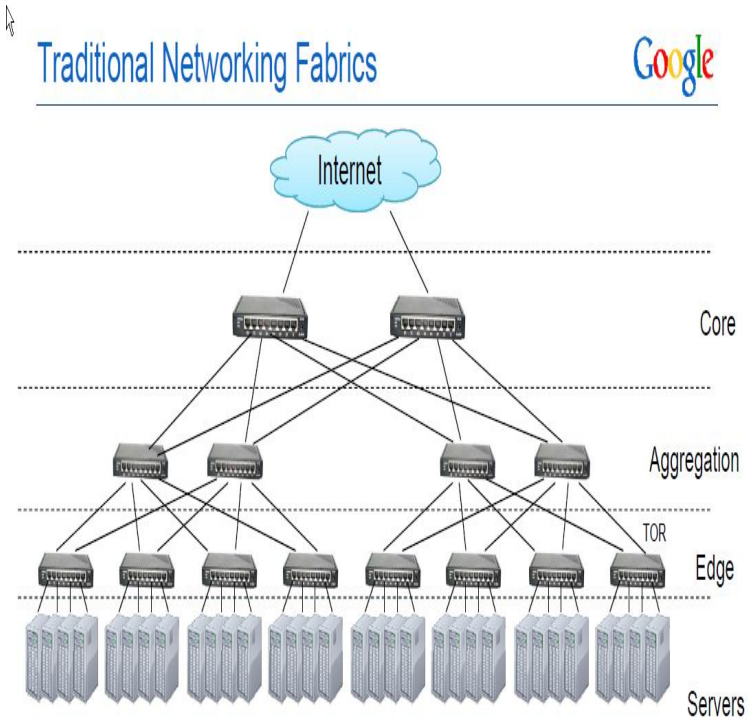


Figure 12: Network topology

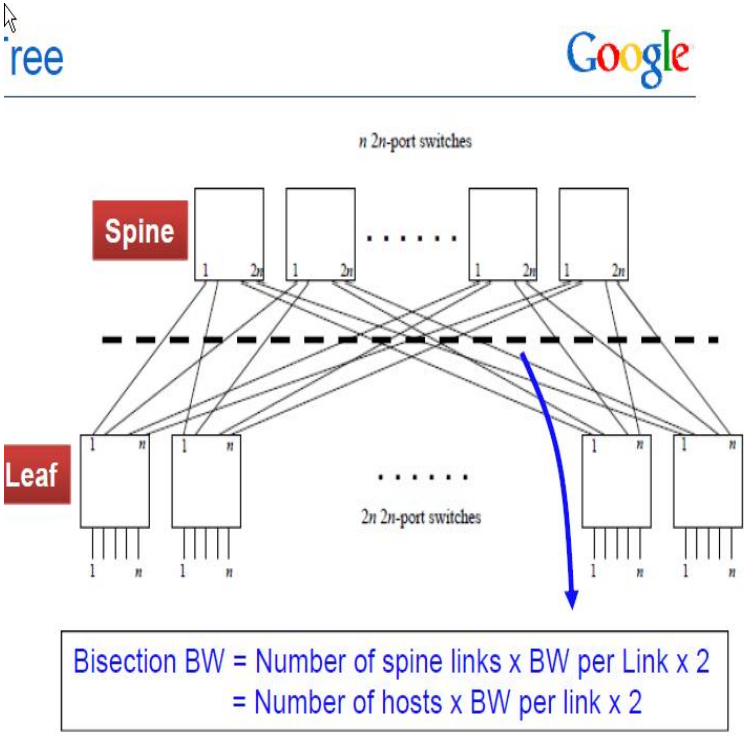


Figure 13: Diagram

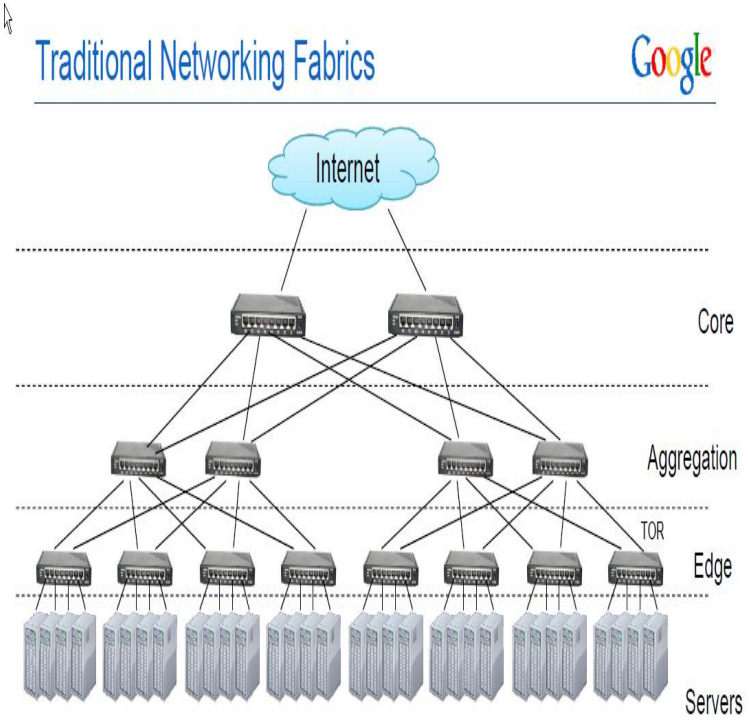


Figure 14: Graphical View