BIG DATA

An Introduction To The Fields Of Data Engineering, Development And Architecture Of Data-Intensive Applications.

—Winter Semester 2018 —

Munich, July 22, 2018

Marcel Mittelstädt
Mail: mittelstaedtmarcel@googlemail.com
Web: www.marcel-mittelstaedt.com

Cooperative State University Baden-Wuerttemberg
DHBW
Duale Hochschule
Baden-Württemberg

Preface

This lecture will give you a brief introduction to so what is called 'Big Data'. We will quickly refresh the basics about databases, data models and data processing you have learned so far and compare those to the distributed world of Big Data.
After that we will take a deep dive into the foundations of distributed data storages and data processing as well as the belonging concepts of reliability, scalability, replication, partitioning, batch and stream processing.
Later on we will take a look at the most common used software and frameworks (mostly the hadoop ecosystem).
At the end, as you know the basic concepts and you are able to setup and work with distributed environments and huge data sets, there will be a short introduction to data science.

At the end of each lesson, there will be some hands-on exercises, which we will start together and which have to be finished till the next week. This lecture will only be about 36 hours in 12 weeks (1 slot each week), which is very little time to cover such an extensive topic. So pay close attention and if you can't keep up, feel free to ask questions at the end of each lesson.

You can find:

- this **script** (and LaTeX-sources),
- **slides** presented within the lecture,
- **excercises** and **solutions**,
- **docker images**, **scripts** as well as **sample data sets**

here:

```
https://github.com/marcelmittelstaedt/BigData
```

You can just download everything directly or install `git` and get everything by using:

```
git clone https://github.com/marcelmittelstaedt/BigData.git
```

and

```
git pull
```

to get the most recent version.

If you find any mistakes or misspellings feel free to send me a mail (**mittelstaedtmarcel@googlemail.com**) or if you are able to, commit a push request.

One last point, as you may have noticed, Microsoft as well as other commercial vendors successfully fail at developing and providing adequate solutions for highly data-driven applications, so almost any software or framework you will encouter during this lecture or later, will be open-source and only be runnable on a UNIX-based operating system. Either you are already familiar with UNIX (lucky you), otherwise you will learn something valuable that will improve your life.

> *"Microsoft is not the answer. Microsoft is the question. NO is the answer."*
>
> — Erik Naggum ✝ , *Philantropist and Developer (Emacs, Lisp and SGML)*

Contents

1 Introduction To Big Data

> *"Big Data is like teenage sex: everyone talks about it, nobody really knows how to do it, everyone thinks everyone else is doing it, so everyone claims they are doing it."*
>
> — Dan Ariely, *Professor of Psychology and Behavioral Economics, Duke University*

1.1 Definition of Big Data

Lorem Ipsum

1.2 Challenges

Lorem Ipsum

1.3 Use Cases

Lorem Ipsum

1.4 Dissociation Datawarehousing

Lorem Ipsum

2 Foundation Of Distributed Data Systems

"I'm not telling you it's going to be easy - I'm telling you it's going to be worth it."
— Arthur L. Williams Jr., *Founder of Primerica Financial Services*

In this chapter we will go trough the foundation of data systems, requirements and concepts which apply to any (data-driven) system. This covers in particular following topics:

- Section 2.1 Requirements of **Data-Driven Systems**, e.g.
  - 2.1.1 Scalability,
  - 2.1.2 Reliability,
  - 2.1.3 Maintainability.
- Section 2.2 **Storage Concepts** for databases
- Section 2.3 **Data Models And Access** concepts
- Section 2.4 **Challenges Of Distributed Data Systems**, e.g.
  - 2.4.1 Partitioning,
  - 2.4.2 Replication,
  - 2.4.3 Transactions,
  - 2.4.4 Consistency amongst others.

At the end you will have a basic understanding about the difference between common and distributed systems and databases, the basic concepts of each of them and which one theoretically fits best to solve a certain problem. A more hands-on deep-dive into related software, frameworks as well as specific problems and use cases will be demonstrated later in chapter 4 Software and Frameworks.

2.1 Data-Driven Systems

When we think about data-driven systems, we mostly think about the same requirements we expect of any other data system we already know:

- **Data Storage**: We need to store data and also need to be able to find it again later (*database*).
- **Data Querying**: We need to be able to query and filter data efficiently in certain kinds of ways (*transaction and indices*).
- **Retention and Performance**: We want results fast, especially of expensive read operations (*caching*).
- **Data Processsing**: We want to be able to process a huge amount of data (*batch processing*) as well as process data asynchronously (*stream processing*).

This sounds quite obvious, but remember those requirements are still the same as for the first database CODASYL[1] back in the 1960's. Even though there are and have been a lot of databases back in time, each of them with a diverse purpose and different approaches to solve e.g. indexing or caching - all of them still match those same requirements. Certainly those data systems evolved much further, especially within the last years, you may noticed:

- Relational Databases being able to handle NoSQL data (e.g. even "retirees" like IBM DB2[2] or Oracle[3]) as well as NoSQL databases being able to handle traditional SQL (e.g. ToroDB[4]) or
- databases becoming message queues (e.g. RethinkDB[5] or Redis[6][7]) and the other way around message queueing systems become databases (e.g. Apache Kafka[8]).

[1] https://en.wikipedia.org/wiki/CODASYL

[2] (IBM18), https://www.ibm.com/support/knowledgecenter/en/SSEPEK_11.0.0/json/src/tpc/db2z_jsonfunctions.html

[3] (ORC18), https://docs.oracle.com/database/121/ADXDB/json.htm

[4] (TOR18), https://www.torodb.com

[5] (RDB18), https://rethinkdb.com/docs/changefeeds/

[6] (RUD18), https://redis.io/commands/rpoplpush

[7] (Joh14), see 5. Adopting Redis for Application Data

[8] (KFK18), https://kafka.apache.org/10/documentation/streams/developer-guide/interactive-queries.html

As you can see, boundaries between traditional databases and data-driven applications get blurred and in the same way more diversified. There is no one-size-fits-all solution, e.g. like you can find back in the past in the 1990's or early 2000s. At that time monolithic single-, 2 and 3-tier, architectures were state-of-the-art (see Figure 2.1 left-hand side).
Usually the **database layer** was represented by a data store like MySQL, Oracle, DB2 or even just files containing data stored on the local disk.
The **application layer** was usually a monolithic application developed in languages like PHP, Perl, C++ or Java and running on a web- or application server (e.g. Apache HTTP Server or IBM WebSphere).
And last but not least the **client layer**: a web browser like nowadays.
If you take a look at Figure 2.2 on page 6 you can see an example of this time

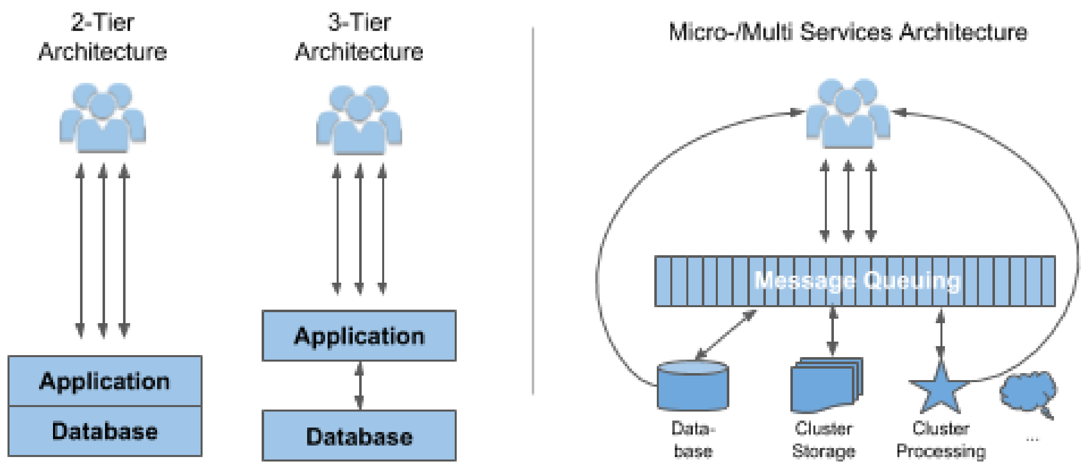


Figure 2.1: Schema - Application Architectures

you may know: ebay.com. They have used the classical 3-tier architecture as well: Oracle as the database running on Solaris as OS[9], C++ as application code running on the Microsoft IIS[10] web server. As you can already guess, this architecture won't scale very well today, in fact the only way to scale this application was to upgrade the single server (*scale up vertically*), in case of ebay.com they once switched from

[9]OS, Operating System
[10](IIS18), *Microsoft Internet Information Services, an extensible web server created by Microsoft for use with the Windows NT family.*

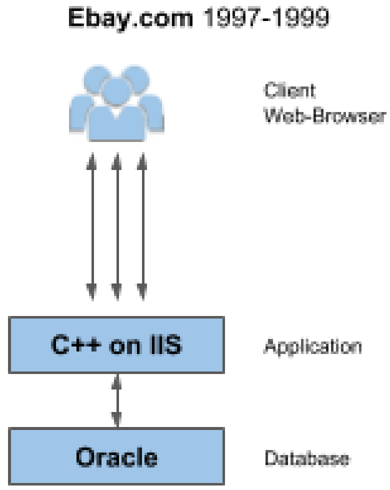


Figure 2.2: Schema - Architecture Ebay.com 1997-1999

commodity hardware to a very pricey mainframe server (Sun Enterprise E10000[11]) to buy some time. But as you may have noticed there are much more ovious issues, e.g. if you think about:

- **Redundancy** does not exist at all (if the database itself or it's server suffers an outage the whole system will be unavailable.
- **Extensability** is not existent, the system is only able to scale up vertically and if this is needed, a downtime is inevitable, as no part of the data system is neither replicated nor virtualized.
- **Maintainablity** is also very limited as any maintenance of the database will require a certain amount of time in which the application will be unavailable.

But we will dicuss this later in the following chapters.

The previously mentioned issues are already sufficient reason but also the increasing amount of data as well as required features of data systems these days (becoming more diversified in the same way) make it unfeasible to rely on a single tool. Instead each functionality is usually broken down into parts which can be done efficiently by suitable tools which are sticked together within the applications itself. This could probably look like as you can see in Figure 2.1 (right-hand side) on page 5, but

[11](SP06), slide 11

that's just one plain example of many other.

Instead of having one single-purpose data store, there are several tied together, each one of them to fulfill it's specific part within the whole data system but all of them tied together as one application.
As you can see one part of the data system could be: a **database** (like you saw in Figure 2.2 on page 6), e.g. to store and serve:

- user data (e.g. in case of an application with login)
- product data (e.g. in case the applications is a web shop)
- user generated content (e.g. in case the application is a newspage, blog or forum)

Another part could be **cluster storage** like Hadoop, which could:

- keep a complete history of all raw data (e.g. page requests of a website or measured values of a sensor)
- serve for batch processing (e.g. crunching the whole history of data, which is impossible for a single database, as it couldn't even save the whole data and certainly wouldn't be able to process it later on)
- serve for analytic and reporting purposes (e.g. reports of how many people have visited the website within the last year based on the raw data)

Also frequently seen, an analytical **cluster processing engine**, e.g. Spark or Flink to:

- process data gathered in real-time (e.g. every page request of a website) for analytical purposes
- use processed data, to run data science models on it (e.g. to serve targeted advertisements or customized content to a user on a website, based on his last page requests, browser user-agent or device)
- ...

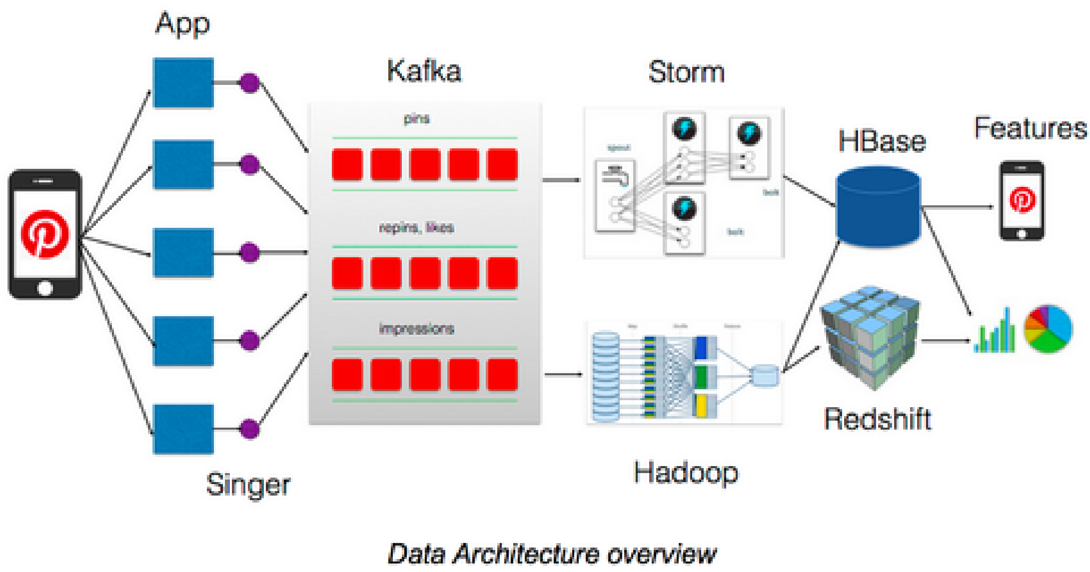


Figure 2.3: Schema - Architecture pinterest.com

If you take a look at Figure 2.3, you can see a comparable data system architecture, implemented by pinterest.com[12]. Redis as a database on top of the hadoop cluster storage to serve for analytical purposes (e.g. ad serving of pinterest's adbuyers) or HBase on top of Hadoop cluster storage and Storm to serve features for the actual end-user of pinterest.com.

But we need to take care here: by creating new and more complex data systems from special purpose data systems, complexity is growing with it. How to ensure the system is avalaible with a reliable performance if something crashes? How to make sure data remains consistent and complete if things go wrong? How to scale the data system to be able to handle increased load?...
There are many apects which are crucial and influence the architecture of a data system like regulary constraints like data security, location of servers, SLA's[13] or

[12](PIN18), Architecture of Giants: Data Stacks at Facebook, Netflix, Airbnb, and Pinterest
[13](WKS18), *Service Level Agreement, a commitment between a service provider and a client. Particular aspects of the service – quality, availability, responsibilities – are agreed between the service provider and the service user.*

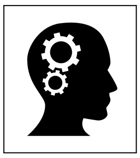
existing devlopment and operation skills - which very much depend on the specific situation.
Within the next chapters we will focus on the aspects which must be taken into account by any data system:

- Scalability (Chapter 2.1.1),
- Reliability (Chapter 2.1.2),
- Maintainability (Chapter 2.1.3).

As many people and companies usually mess around with those terms, firstly we will develop a clear understanding on what they mean and later on take a closer look on how to apply algorithms, development and architectures to fulfill them appropriately.

2.1.1 Scalability

As is evident from the introduction of this chapter: the fact that a system is working reliable today doesn't mean it will necessarily work reliable in the future. The data system of ebay.com in 1999 was maxed out at handling **50.000** active listings[14], imagine how the system would behave today at handling **1 billion** active listings[15]. Obviously handling increased load (e.g. a larger amount of data needed to store or request to handle) is a major factor of a scalable data system.



Scalability is the capability of data system to handle a growing amount of load (e.g. a larger amount of data needed to store or requests to handle). A data system is considered scalable if its capable of increasing it's total through-/output under an increased load when resources (typically hardware) are added.

Note that, "*scalability*" isn't a binary tag that could be attached to a data system. It's pointless to say "*a data system is scalable*" as well as "*a data system is not scalable*", in either way you must think about "*If the load of a data system grows in a certain way, what are the options on the table for coping with the growth?*" and

[14](SP06), slide 9
[15](EBA18)

"*How can we add ressources (hardware) to be able to handle the additional load?*". Therefore we will discuss the parameters and definition of **Load** (Section 2.1.1.1) and **Performance** (Section 2.1.1.2) within the next section as well as **Approaches** for coping with load to achieve a certain performance (Section 2.1.1.3).

2.1.1.1 Load

To get an idea what *load* of a data system actually means, how it could be described an measured, we will take a nother look at the example of ebay.com discussed so far. If we take a look back at the architecure of ebay at 1997-1999 (Figure 2.2 on page 6) we can already guess with increasing load (page requests to certain items listed on ebay.com and in this way calls to the database through the application) will reach its limit at the maximum amount of read requests the oracle database can serve. As the application tier (web server) has already been scaled horizontally to multiple nodes, the oracle database server reached its limit of physical growth in November 1999.

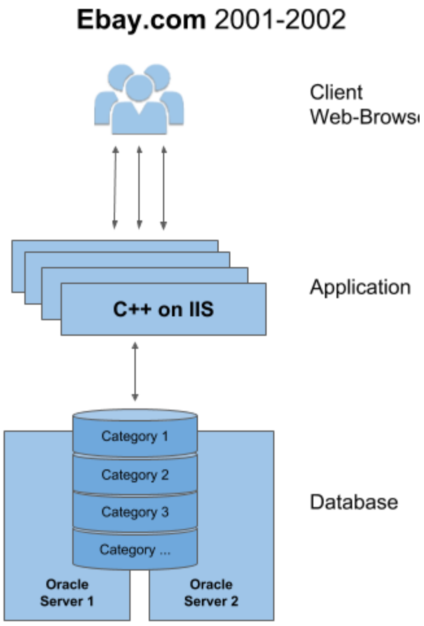
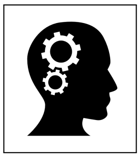


Figure 2.4: Schema - Ebay Architecture April 2001 - December 2002

So ebay added an additional server to not just elmininate the SPOF[16] to be able to failover but also they have splitted the database to be able to logically partition it into separate instances and in this way be able to scale horizontally. This was achieved in 2001 by splitting items by categories, as you can see in Figure 2.4 on page 10. In this simple way it was possible to distribute the load (mostly page requests for items) in an "equal" way to different physical nodes. Later on they segmented whole databases into functional areas like hosts for item, user, account or transaction data as well as partitioned the data by typical usage characteristics to scale horizontally.

They obviously did furher optimizations at the whole data system to be able to cope with the increasing load, like disabling transactions, moving CPU-intensive[17] work to the application tier (e.g. joins, referential integrity or sorting), extensive use of prepared statements...but as this techniques are not mainly specific for distributed systems and some of them not even recommened nowadays, we won't focuse on them within this lecture.

In the example of ebay.com, requests per item and category could be a valuable *load parameter* for discussing scalability, since it determines the database requests per data record and partition - as proven by the the data system structure of ebay at 2002 as we see. Your or other data systems you have seen so far most likely have very different characteristics, but you can apply similar principles to reason about their load.



The **load** of a data system is a measurement of the amount of computational work it performs (depending on the architecture in-place), e.g. the number of (concurrent) reads from a data storage, writes to a data storage or the ratio between reads and writes. The maximum load is defined by the weakest part of the architecture (=*bottleneck*).

[16](WPS18), Single Point Of Failure

[17]CPU, a processor or processing unit is an electronic circuit which performs operations on some external data source, usually memory or some other data stream is called central processing unit

2.1.1.2 Performance

Now that we have described what *load* of a data system means as well as what *load parameters* could be, we will examine more closely what happens when the load increases. Usually there are two important cases you need to think about while developing data systems:

- The load parameter increases, but all ressources (e.g. number of server, CPU or memory) stay the same - how is the performance of the data system affected?
- The load parameter increases - how much do you need to increase the ressources (e.g. number of server, CPU or memory) to keep the performance stay the same?

But how to answer them? Therefore we need performance numbers. In case of data systems measurement, methods usually are **throughput** (number of records that can be handled), e.g.:

- read/writes per second (in case of MongoDB up to 100.000 read/writes per second)
- messages processed (in case of Apache Kafka and Linkedin more than 2 million records per second on just 3 nodes)
- data processed (in case of Apache Hadoop and MapReduce terabytes of data within several seconds)

or if your buidling a data-system which works as the backend of a end-user facing application like a website, it's more about the **response time**, which means the time between sending a request and receiving the response. For instance if we think about the example of ebay.com within the previous chapters, as of 2016 they had 1 billion items accessible at any time, needed to serve 2 billion page requests each day and had to fulfill each of them within fractions of a second.

Regardless of throughput or response time - if we think about performance we don't think about a single number, but a distribution of values that we can measure. If you will repeat the same page request, read or writes on the same data system:

response time and throughput will inevitable vary somehow. There are simply too many factors you usually cannnot contain:

- network issues (e.g. latency or TCP packet loss and retransmission)
- os issues (e.g. a page faults, context switches or running background processes)
- physical issues (e.g. a damaged disk/ssd or overheating of a CPU and, associated therwith a decreased processing power)

Therfore it is common to use an average for measuring throughput or response times. *Average* doesn't refer to a particular formula, we will briefly discuss 3 that are commonly used:

- **arithmetic mean**[18] (easy to calculate but ignores ratios and is highly affected by statistical outliers, so it cannot tell you how many requests, reador writes actually have had a worse performance)
- **median**[19] (easy to calculate and less distorted by outliers)
- **percentiles**[20] (easy to calculate, not distorted by outliers)

As you can guess, evaluating symmetric distributions with no outliers, *arithmetic mean* will be the best choice, but as we are looking at performance parameters like throughput and response times, symmetric distribution is a whishful thinking. More usually throughput and response times will result in skewed distributions, so *median* seems to be the better choice, as it doesn't ignore ratio and outliers completely. For instance if you calculate the median for latency (y-axis) of reads in a given timeframe (x-axis) from a data system as illustrated in figure 2.5 on page 14. You can see a small peak at *07:08:00* but it still looks fine as the median response time is < 100ms. But what about the green graph (*95th percentile*)? That's the main reason why using percentiles is pretty common, especially the 95th, 99th or 99.9th percentile (abbreviated *p95*, *p99*, *p999*) is frequently used in SLA's[21]. Percentiles define thresholds at which 95%, 99% or 99,9% of requests, reads or writes are beneath that threshold. Looking back at figure 2.5 this would mean that 95%

[18] Sum of values divided by the number of values.

[19] A median separates the higher half of values from the lower half.

[20] A measure used for indicating a certain percentage of scores falls below that measure.

[21] (WKS18), *Service Level Agreement, a commitment between a service provider and a client.*

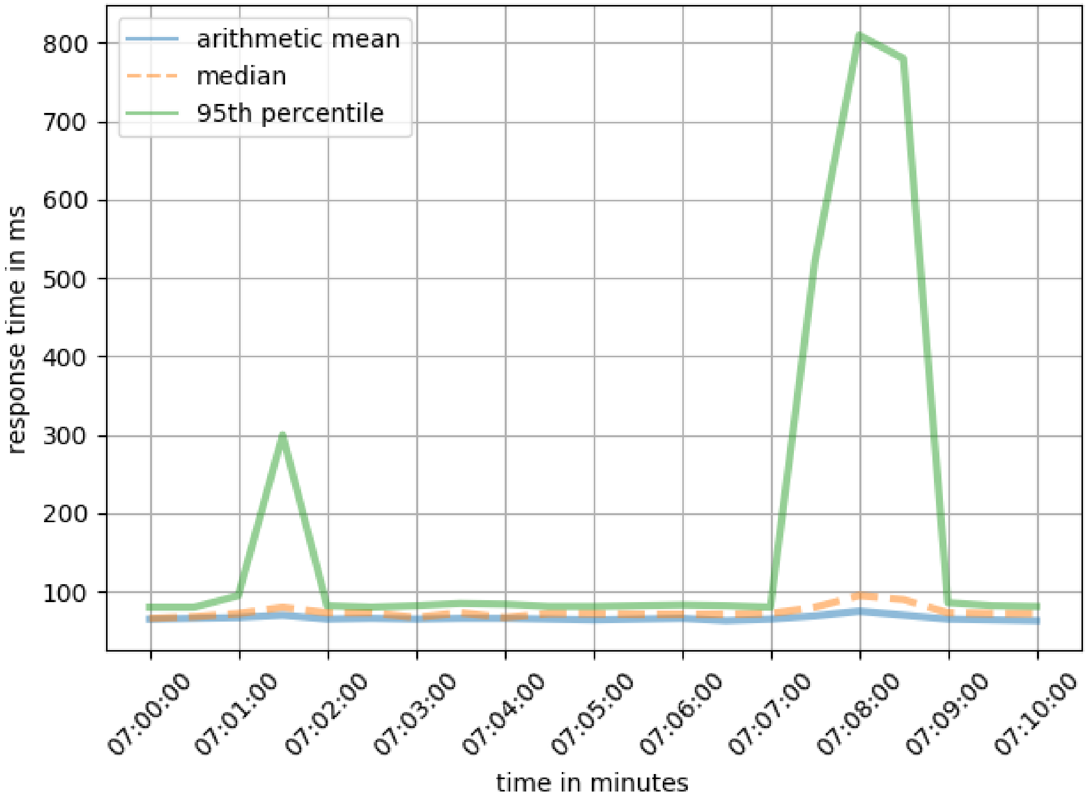
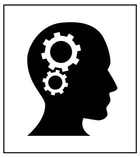


Figure 2.5: Schema - Arithmetic Mean, Median and Percentiles Example

of all requests, reads or writes done by user are faster or equal 810ms and 5% will result in a response time > 810 ms, which in case of a customer facing data system could mean: 5% unsatisfied users and in this way probably a loss of possible leads (e.g. purchases on a webshop or subscriptions for a video streaming platform) and ultimately loss of revenue.

So why don't use 99,9th percentile every time as it is the best? This is a major cost factor, optimizing the last percentiles gets really expensive, especially as this usually involves a lot of hardware redundancy as well as eliminating factors outside of your control. At a certain point costs will be bigger than the benefits, so you need to make a trade-off.



Performance of a data system is defined by system throughput and response time, e.g. number of transactions (like read/write operations), processed records (like aggregations for analytical purposes) or even system commands (like an *update statistics* or rebalancing of several data nodes) under a given workload and for a specific timeframe. It usually depends on a variety of influencable as well as uninfluencable factors of the system itself, like network latency, a page fault or damaged disk.

2.1.1.3 Approaches For Scaling

Now that we are familiar with describing *load* and measuring *perfromance* we can answer the question: how to ensure performance, even if the load increases? As we have seen by the example of ebay.com within the previous chapters, a system which is able to cope with a load, won't be able to handle 10 times of that load. This needs us to think about the architecture right at the beginning as well as each time the load significantly increases. As we have learned there are two ways to scale an architecture of a data-system:

- **scale up/vertical** - replace a server by a more powerful one
- **scale out/horizontal** - distribute the load towards multiple server instead of one

A data-system running on a single server is easier to develop, as you can neglect a lot of factors that are specific to distributed systems (e.g. replication, partitioning or transactions and consistency across nodes), but more powerful machines are also more expensive and at a certain point you will reach the physical limit as ebay.com did.
A distributed data-system will require more development, test and operational effort as well as result in complexity but servers will be much cheaper as you make use of less powerful machines (*"commodity" hardware*) and you are able to bypass the inevitable physical limit of a single server.
In practise you won't choose one pattern only (scale up or out) as well-working architectures need a carefully chosen mixture of both approaches, e.g. it doesn't

make sense to make use of a lot of poorly powered servers (like a Raspberry Pi) instead of some more powerful machines in terms of unnecessary costs, network and software complexity. As you may guess, there is no one-size-fits-all solution and architecture for scalabale data-systems as the requirements are highly specific to each data-system itself. The *load parameter* may be strongly influenced by:

- the **volume of data** to store
- the **number of read or write operations**
- the required **throughput** or **response time**
- the **structure of the data** and **how it's accessed** (e.g. relational, document-oriented, graph)
- and many more.

Right now it shall be sufficient that you know the basics concepts of scalability, later within this lecture, we will make use of it when looking at distributed data storage and processing as well as related software and frameworks. At the end you should be able to apply those concepts to any data-system and be able to make reasonable decisions in terms of scalability.

2.1.2 Reliability

In general *reliability* represents the probability that something/someone will perform a required function without failure under stated conditions for a period of time, e.g. a test will be reliable when it gives the same repeated result under the same conditions.
Or more pragmatic: *something works correctly even if things go wrong.*

So what are the *faults*, mentioned in the previous defintions that we need to anticipate, about?

2.1.2.1 Hardware Faults

Obviously any hardware produced has a certain lifespan, buth that's not the only reason for hardware faults. If you're talking with operators of data centers, they will provide you with a broad list of common as well ase spine-chilling causes for hardware faults as:

- broken HDDs[22] or SSDs[23]
- faulty RAM[24] or CPUs[25]
- broken power adapters, switches or whole network outages
- unplugged network cables or even connected to the wrong port
- and many many more.

As this seems pretty unlikely at first sight - it's definitely not. For instance let's

[22] HDD, a hard disk drive is a non-volatile computer storage device containing magnetic disks or platters rotating at high speeds.

[23] SSD, a solid-state drive is a nonvolatile storage device that stores persistent data on solid-state flash memory.

[24] RAM, a Random Access Memory is the hardware in a computing device where the operating system, application programs and data in current use are kept so they can be quickly reached by the device's processor.

[25] CPU, a processor or processing unit is an electronic circuit which performs operations on some external data source, usually memory or some other data stream is called central processing unit

have a look at hard drives, especially in distributed data-systems you will have a lot of them. If you think about Apache Hadoop, you usually use low-class server (*"commodity" hardware*), e.g. *ProLiant DL380 Gen10 Server* as they provide a good ratio of:

- computing power (CPU) / Storage (HDD),
- rack space (2 RU[26]) / storage and
- benefit/cost.

Each of this servers can store 19 HDDs, if you build a hadoop cluster with those servers, e.g. with about 100 nodes, this means 1,900 HDDs. Based on a regularly study by BackBlaze[27] (a big data storage center provider like Amazon AWS) with a set of 82,516 HDDs, the average annual failure rate is about 2.11%. Regarding our previous Hadoop example containing 1.900 HDDs, we can suppose that nearly any week a HDD will fail. If we would make use of the particular HDD model *Seagate ST4000DX00* with a failure rate of 35,88% (also mentioned within the study) this would mean nearly 2 HDDs would fail each day.

In single server data systems it is possible to mitigate those problems by adding redundancy to individual hardware parts to minimize the failure rate of the whole system to a point where a failure is very unlikely, as at any time a redundant part can take over. This could mean, dual power adapters (like used by the previous mentioned *ProLiant DL380 Gen10 Server*), RAID[28] configurations or hot-swappable CPUs. As data volumes and computing demand increases, data-systems need to be distributed among several servers, which proportionally increases the rate of hardware faults and system failures, like discussed above regarding HDD faults. Therefore distributed data systems need to be able to tolerate the loss of entire machines, requiring software to be fault-tolerance additionally to hardware redundancy. But those distributed data systems have further advantages, a system that tolerates

[26](WPR18), Rack Unit is a unit of measure defined as 44.50 millimetres (1.75 in). It is most frequently used as a measurement of the overall height of rack frames.

[27](HDD17), https://www.backblaze.com/blog/hard-drive-failure-rates-q1-2017/

[28]RAID, is a data storage virtualization technology that combines multiple physical disk drive components into one or more logical units for the purposes of data redundancy, performance improvement, or both.

failure of single machines can be restarted, patched, updated (*rolling-upgrades*) or maintained - one node at a time - without a downtime of the whole data-system.

2.1.2.2 Software Faults

When we talk about software faults in terms of distributed data-systems, we don't talk about usual bugs but rather faults which affect the whole data-system integrity and reliability. Such faults are harder to anticipate than usual bugs of single-server applications, as they usually tend to be caused by the environment (e.g. multiple servers, network, dependencies, special and unusual circumstances), are difficult to test, and are even worse in their result as they can cause a failure of the whole data-system. Examples could be:

- a runaway and/or zombie process that extensively used up some shared ressource (e.g. network, CPU, RAM, disk space)
- a software bug causing the whole cluster to fail (e.g. the Hadoop Ressource Manager YARN[29] once had a bug[30], that if you removed a cgroup (*control group*) under some circumstances (*race conditions*) a kernel panic and in this way a failure of multiple server was caused
- a service the whole data-system depends on slows done, becomes unresponsive or fails
- cascading failures (e.g. one server of the data-system fails due to heavy network traffic, causing the other servers to take over, in this way increasing network traffic for them too and finally all server will fail)

As you can see, most of the reasons for software faults are caused by assumptions about the environment that may not be true at some time and at some special circumstances. To avoid suffering those issues you need to carefully think about assumptions and interactions within the distributed data-system, you will need a lot of measuring, monitoring and you will do a lot of analyzing of the system behaviour in any special circumstance as well as testing even with forcing some servers of the system to crash.

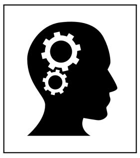
[29] YARN, Yet Another Resource Negotiator
[30] (YAR18), https://issues.apache.org/jira/browse/YARN-2809

2.1.2.3 Human Faults

We have been talking a lot about reliability so far, but what about the most unreliable factor: humans. We will briefly discuss some approaches to make a data-system reliable in terms of unreliable humans:

- decouple places where people make the most failures from places they can cause failures, e.g. using production and development environments or providing interfaces or frameworks for an API instead of direct API access
- use extensive testing (e.g. unit test, system tests, integration tests) and automize them
- measuring and monitoring (e.g. performance metrics, error rates) allows to check wether assumptions or constraints are violated at an early stage

To sum up the last chapters: why do wee need reliability? Reliability is not just a major topic for stock exchanges, air traffic control or military. Failures of a data-system can cause data loss, lost productivity or sales loss and therefore huge costs and loss of revenue. There are special circumstances when you may choose to reduce reliability for the sake of time, development effort or operational costs (e.g. *prototyping*), but you need to be very conscious and it's inevitable that at some point in the future you will need to invest the the saved effort, time and costs and probably a multiple of what it would have been before.



Reliability in terms of hardware, software or especially data-systems can be defined as the ability of a system to function as specified and expected. A reliable data-system also detects and tolerates faults due to mistakes of users, hardware or lower parts of the data-system itself as well as ensures the required performance under any expected load.

2.1.3 Maintainability

Maintenance is known as one of the biggest costs at software devlopment and in the same way a very unfamous topic to software engineers. Keeping a system running, investigating failures, fixing bugs, adding new features, adapting it to updates of underlying hard- and software - to name some usual tasks.

A data-system should be designe to minimize effort during maintenance and in this way making it more reliable. We will briefly discuss 3 major topics: *operability*, *simplicity* and *evolvability*.

2.1.3.1 Operability

The main goal of operability should be to make operations easy to keep the system running smoothly, this means making routine tasks easy and enable operations ressources to use their time for important tasks. This can be achieved by:

- good documentation and operational model (a data-system which can be understand easily can be operated more easily)
- transparency (visibility into the data system and runtime behaviour, e.g. by log files or monitoring tools)
- no dependencies between single services or server (allow single server to go down for maintenance tasks, e.g. patches, update or restarts)
- self-healing if possible, but also possibilities to override for operators

2.1.3.2 Simplicity

When you start a development project everything is pretty simple and probably well-documented but later on with multiple developers, features, services and servers, it gets more complex, hard to understand by a developer and especially more difficult to handle by administrators. As a lot of issues caused by this are not specific to data-systems we will focuse on complexity and abstraction, as reducing complexity should be the main goal when devloping distributed data-systems. Making a sys-

tem less complex doesn't require reducing functionaility, it's more about removing unnecessary complexity.
For instance if your data-system is crunching a lot of data for a very plain purpose, like parsing web server log files for analytical purpose to get to know how many people have visited your website - you could do this counting in Java (MapReduce) but this will probably be about 50 lines of code, a lot of libraries, testing and dependencies - making operations more difficult in the same way. If you would do this using Hive on HDFS your done with a one-line SQL statement.

However finding useful abstractions is not that easy and needs a lot of experience, but when you are developing something you should always ask yourself: do I make use of abstraction and will the complexity be at a manageable level?

2.2 Storage Concepts

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

2.3 Data Models And Access

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

2.4 Challenges Of Distributed Data Systems

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

2.4.1 Partitioning

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

2.4.2 Replication

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

2.4.3 Transactions

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

2.4.4 Consistency

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

3 Data Processing On Distributed Systems

Begin at the beginning, the King said gravely, "and go on till you come to the end: then stop."

—Lewis Carroll, *Alice in Wonderland*

3.1 Batch Processing

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

3.2 Micro-Batch Processing

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

3.3 Stream Processing

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

3.4 Message Queuing

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

3.5 ETL and Workflow Automation

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

4 Software and Frameworks

> *Begin at the beginning, the King said gravely, "and go on till you come to the end: then stop."*
>
> —Lewis Carroll, *Alice in Wonderland*

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

5 Data Science

"Big data is not bout the data."
— Gary King, *Harvard University*

5.1 Data Cleaning, Integration and Preparation

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

5.2 Data Visualization

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

5.3 Regression

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

5.4 Classification

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

5.5 Clustering

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

5.6 Association

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

5.7 Neural Networks

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

5.8 DataScience on Distributed Systems

Spark ML PySpark

6 Outlook

“Big data is not bout the data.”

— Gary King, *Harvard University*

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

7 Appendix

Abkürzungsverzeichnis

CPU .... Central Processing Unit
DHBW .... Duale Hochschule Baden-WŸrttemberg
HDD .... Hard Disk Drive
OS .... Operating System
RAID .... Redundant Array of Independent Disks
RAM .... Random Access Memory
RU .... Rack Unit
SLA .... Service Level Agreement
SPOF .... Single Point Of Failure
SSD .... Solid-State Drive
YARN .... Yet Another Resource Negotiator

List of Figures

Auszugverzeichnis

Bibliography

[EBA18] *What we do - eBay is where the world goes to shop, sell, and give.).* https://www.ebayinc.com/our-company/who-we-are/. Version: July 2018, Abruf: 15. 07. 2018 15

[HDD17] *Backblaze HDD Failure Study.* https://www.backblaze.com/blog/hard-drive-failure-rates-q1-2017/. Version: July 2017, Abruf: 20. 07. 2018 27

[IBM18] *IBM DB2 Knowledgebase.* https://www.ibm.com/support/knowledgecenter/en/SSEPEK_11.0.0/json/src/tpc/db2z_jsonfunctions.html. Version: July 2018, Abruf: 13. 07. 2018 2

[IIS18] *Microsoft IIS Homepage.* https://www.iis.net/. Version: July 2018, Abruf: 14. 07. 2018 10

[Joh14] Johanan, Joshua: *Buidling Scalable Apps With Redis And NodeJS.* Packt Publishing, 2014 7

[KFK18] *Apache Kafka User Documentaion.* https://kafka.apache.org/10/documentation/streams/developer-guide/interactive-queries.html. Version: July 2018, Abruf: 13. 07. 2018 8

[ORC18] *IBM DB2 Knowledgebase.* https://docs.oracle.com/database/121/ADXDB/json.htm. Version: July 2018, Abruf: 13. 07. 2018 3

[PIN18] *The eBay Architecture (Randy Shoup and Dan Pritchett).* https://blog.keen.io/architecture-of-giants-data-stacks-at-facebook-netflix-airbnb-and-pinte Version: April 2018, Abruf: 14. 07. 2018 12

[RDB18] *RethinkDB User Documentaion*. https://rethinkdb.com/docs/changefeeds/. Version: July 2018, Abruf: 13. 07. 2018 5

[RUD18] *Redis User Documentaion*. https://redis.io/commands/rpoplpush. Version: July 2018, Abruf: 13. 07. 2018 6

[SP06] Shoup, Randy ; Pritchett, Dan: *The eBay Architecture (Randy Shoup and Dan Pritchett)*. http://www.addsimplicity.com/downloads/eBaySDForum2006-11-29.pdf. Version: November 2006, Abruf: 14. 07. 2018 11, 14

[TOR18] *ToroDB Website*. https://www.torodb.com/. Version: July 2018, Abruf: 13. 07. 2018 4

[WKS18] *Wikipedia - SLA*. https://en.wikipedia.org/wiki/Service-level_agreement. Version: July 2018, Abruf: 13. 07. 2018 13, 21

[WPR18] *Rack Unit*. https://en.wikipedia.org/wiki/Rack_unit. Version: July 2018, Abruf: 20. 07. 2018 26

[WPS18] *Single Point of Failure*. https://de.wikipedia.org/wiki/Single_Point_of_Failure. Version: July 2018, Abruf: 20. 07. 2018 16

[YAR18] *YARN Jira - Ticket YARN-2809*. https://issues.apache.org/jira/browse/YARN-2809. Version: July 2018, Abruf: 20. 07. 2018 30