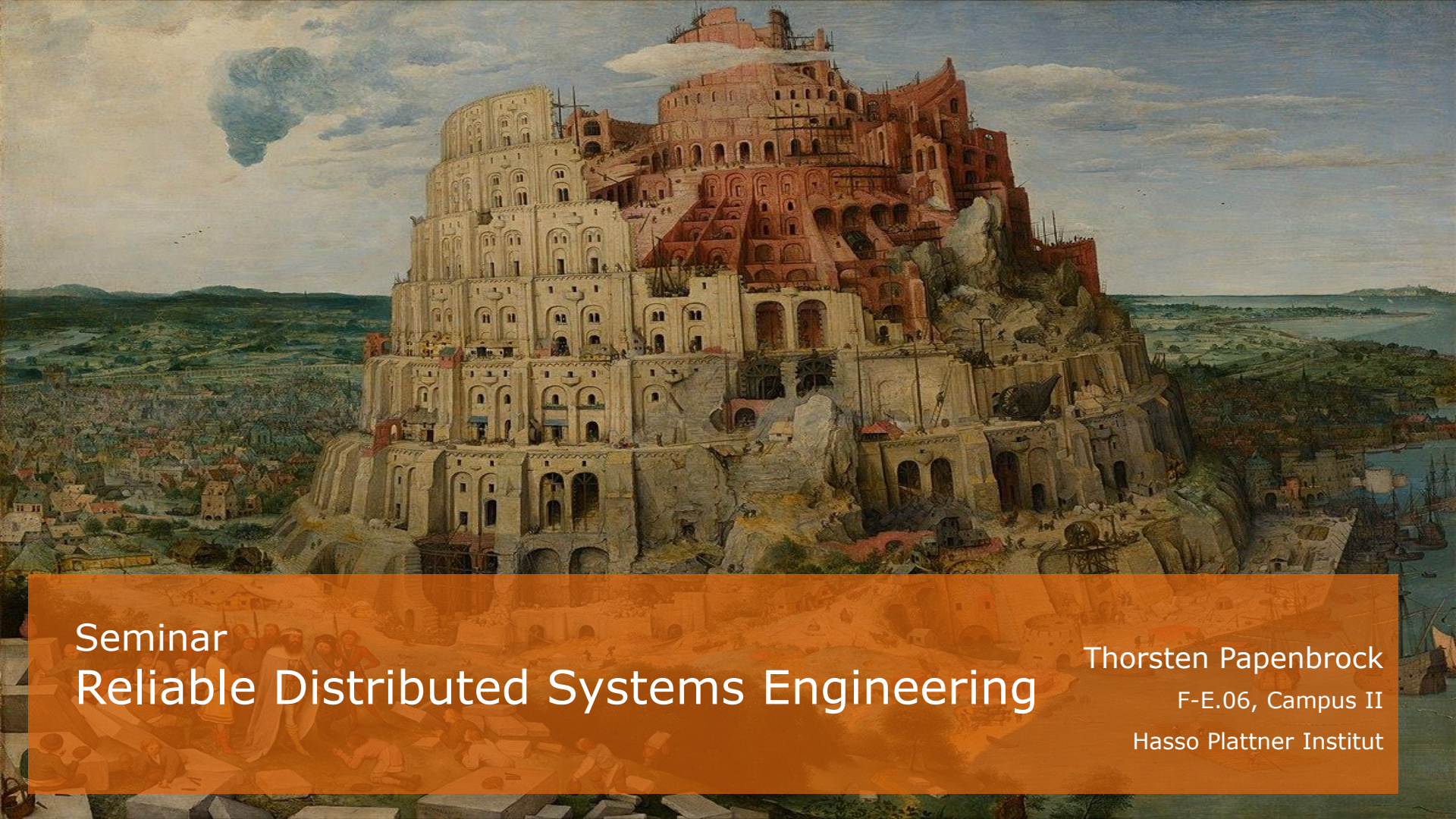


Seminar

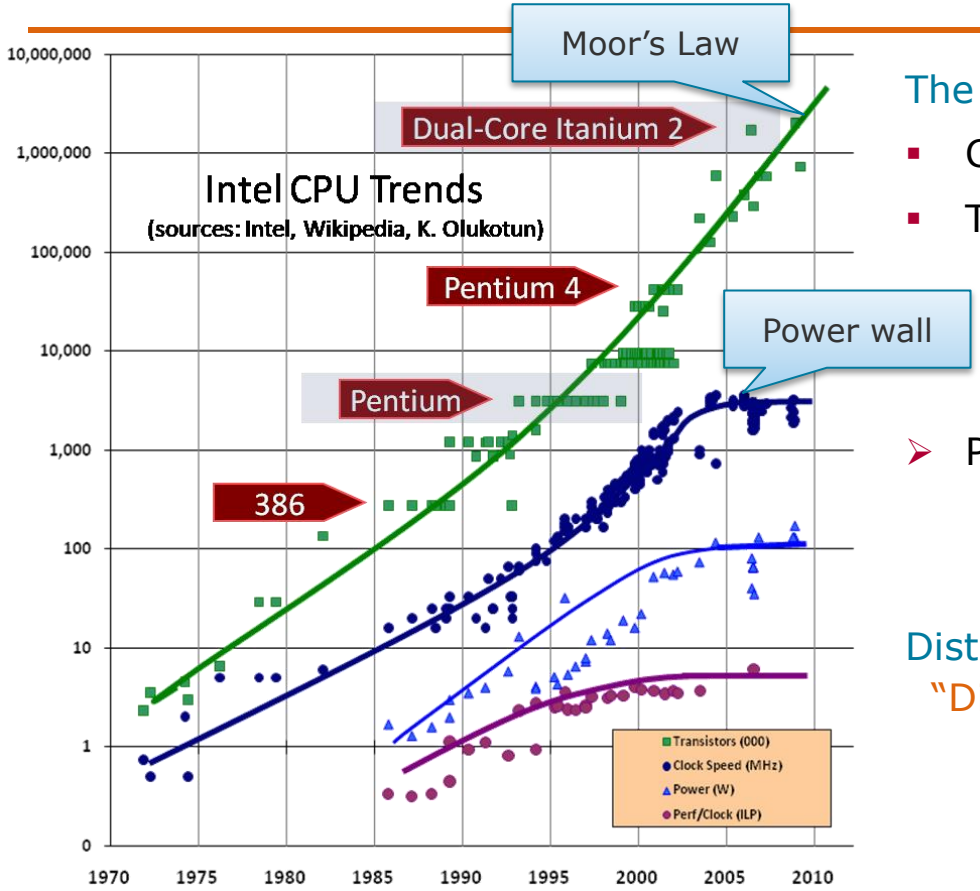
Reliable Distributed Systems Engineering

Thorsten Papenbrock

F-E.06, Campus II

Hasso Plattner Institut

Paradigm Shift in Software-Engineering



The free lunch is over!

- Clock speeds stall
- Transistor numbers still increase
  - Cores in CPUs/GPUs
    CPUs/GPUs in compute nodes,
    compute nodes in clusters
  - Paradigm Shift:
    - Earlier: optimize code for a single thread
    - Now: solve tasks in parallel

Distributed computing

"Distribution of work on (potentially) physically isolated compute nodes"

http://www.gotw.ca/publications/concurrency-ddj.htm

Motivation

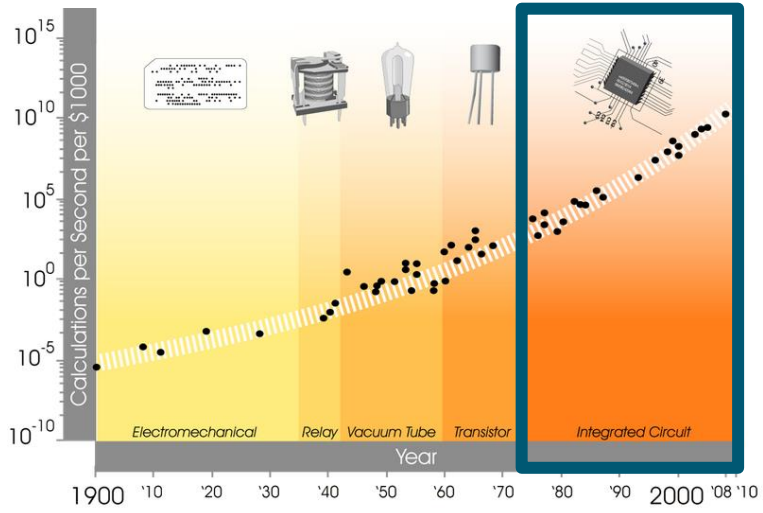
Surpassing Moor's Law

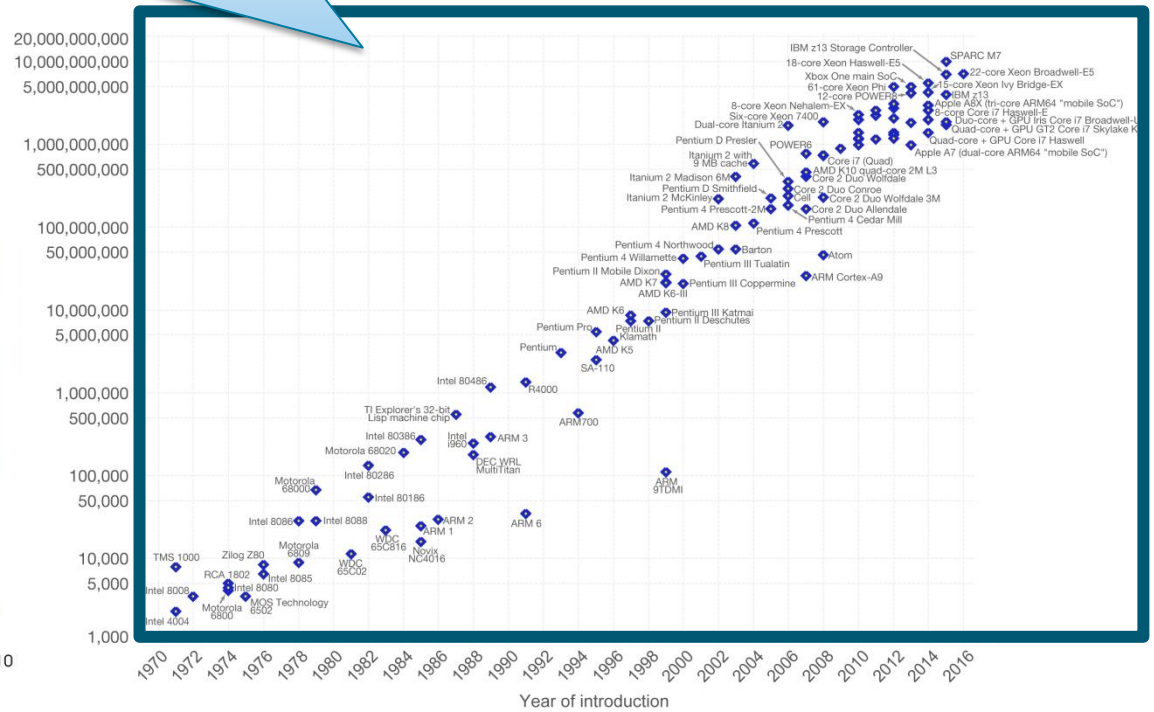
With **distributed machines**, we can already build systems with any number of transistors!

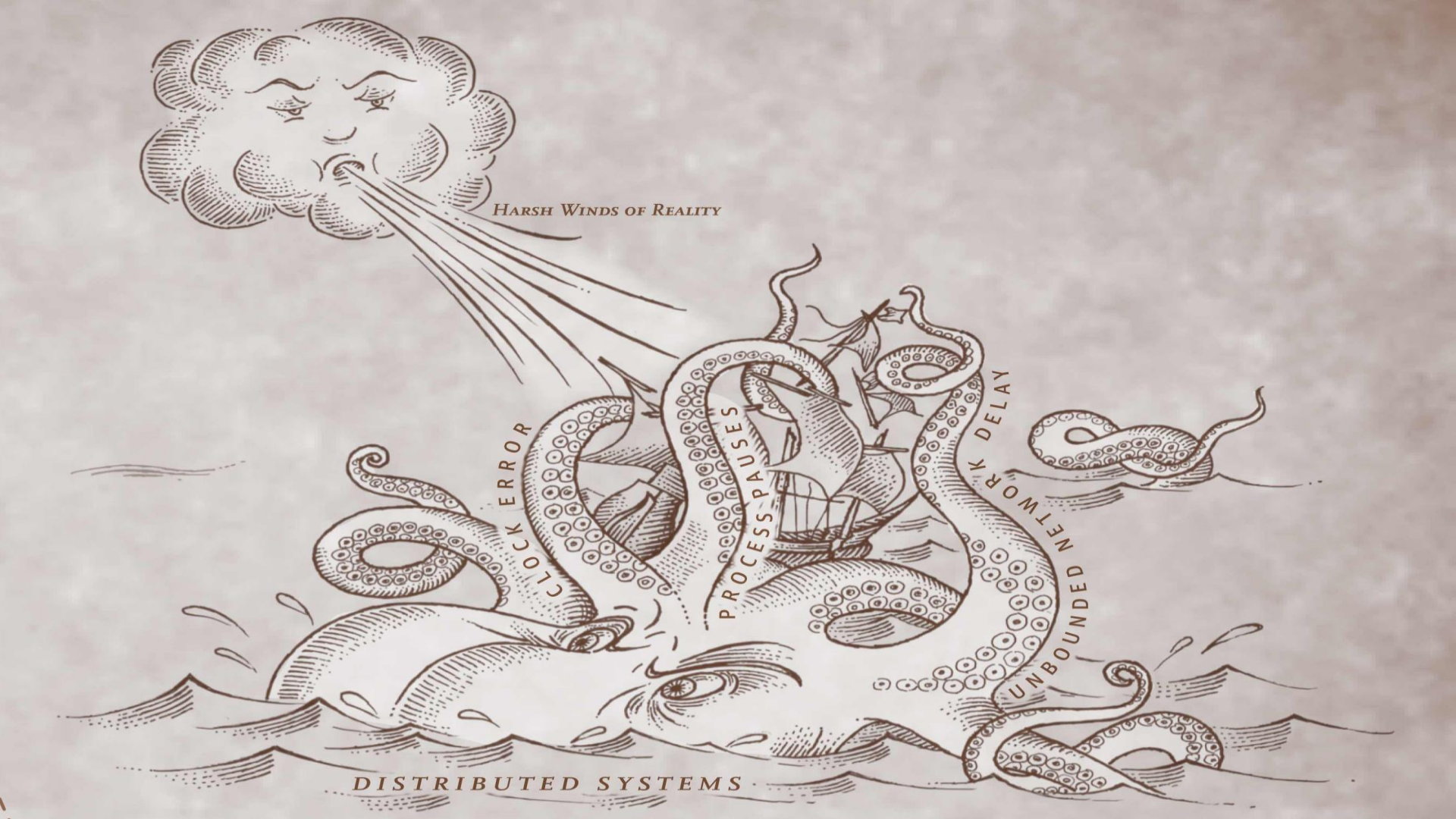
(don't even need to wait for a new processors)

Moore's Law (Observation)

"The number of transistors in integrated circuits doubles approximately every two years"






Harsh Winds of Reality
Clock Error
Process Pauses
Unbounded Network Delay
Distributed Systems

Distributed Computing Challenges

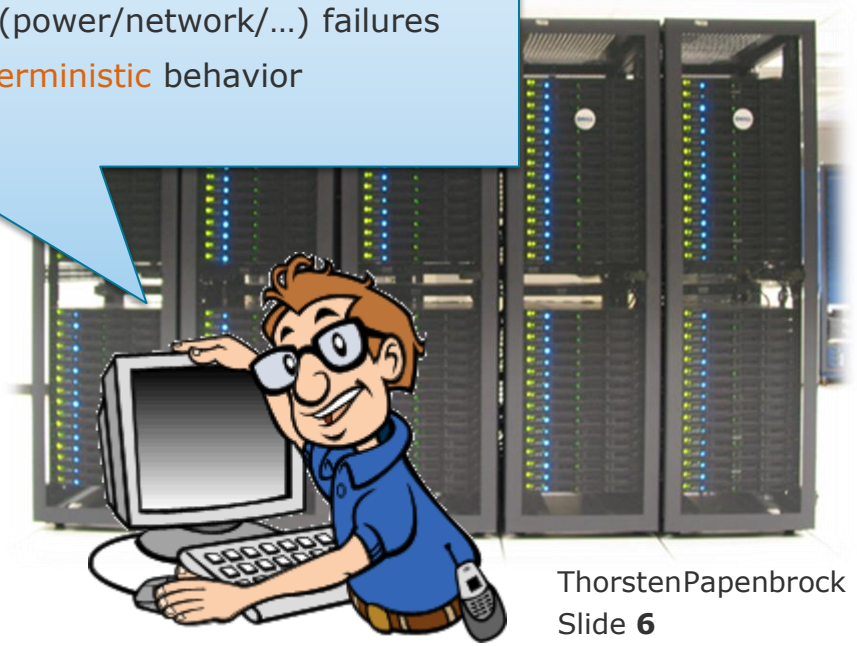
I am facing …

- software bugs
- power failures
- head crashes
- hardware aging
- …

Non-Distributed System Developer

I am facing everything he faces and …

- network faults
- clock deviation
- partial (power/network/…) failures
- nondeterministic behavior
- …



Distributed System Developer

Distributed Computing Challenges

"My system is predictable"

"I can debug easily"

"A well operating system should not have failures"

"I use parallelism whenever necessary"

Non-Distributed System Developer

"My system is predictably unpredictable"

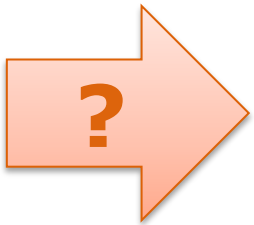
"Debugging is hard"

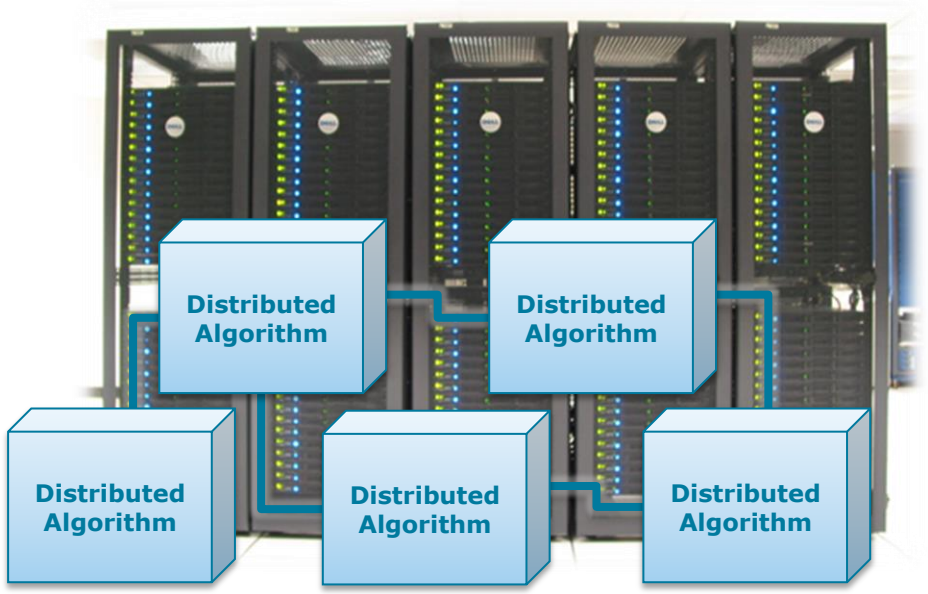
"A well operating system properly deals with its failures"

"Parallelism is my bread and butter"

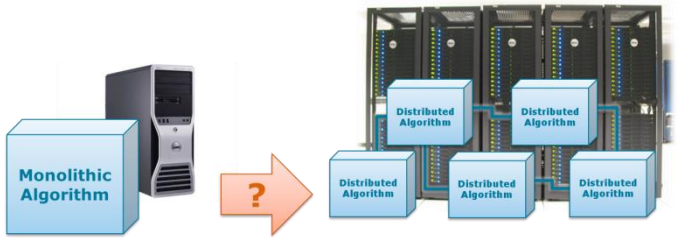
Distributed System Developer

Distributed Computing Challenges





Distributed Computing Challenges

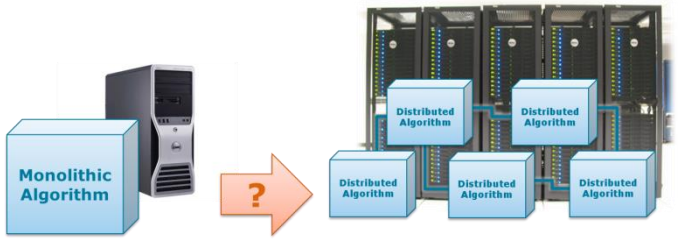


Questions

- How do we change a particular algorithm to efficiently execute on various, independent and heterogeneous computing elements?
- How do we utilize all available resources in an optimal way?
- How does the algorithm deal with the increased error susceptibility of a parallel/distributed system?
- How can the algorithm support elasticity, i.e., sets of computing resources that change at runtime?
- How does a parallel/distributed system control its resource consumption in terms of overall memory and CPU usage?
- How do we start and terminate such systems?
- How do we debug, monitor, and profile them?

Motivation

Distributed Computing Challenges



Questions

? How do we change a particular algorithm to efficiently execute on various, independent and heterogeneous computing elements?

- ✓ How do we utilize all available resources in an optimal way?
- ✓ How does the algorithm deal with the increased error susceptibility of a parallel/distributed system?
- ✓ How can the algorithm support elasticity, i.e., sets of computing resources that change at runtime?
- ✓ How does a parallel/distributed system control its resource consumption in terms of overall memory and CPU usage?
- ✓ How do we start and terminate such systems?
- ✓ How do we debug, monitor, and profile them?

hadoop MapReduce

APACHE STORM™

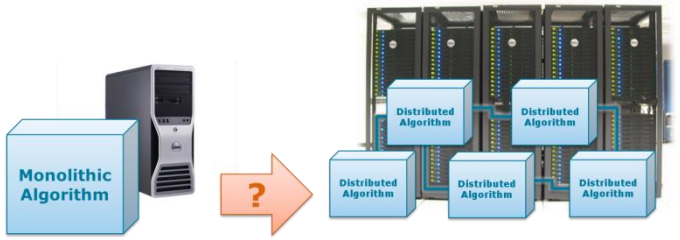
APACHE Spark™

Apache Flink

TEZ

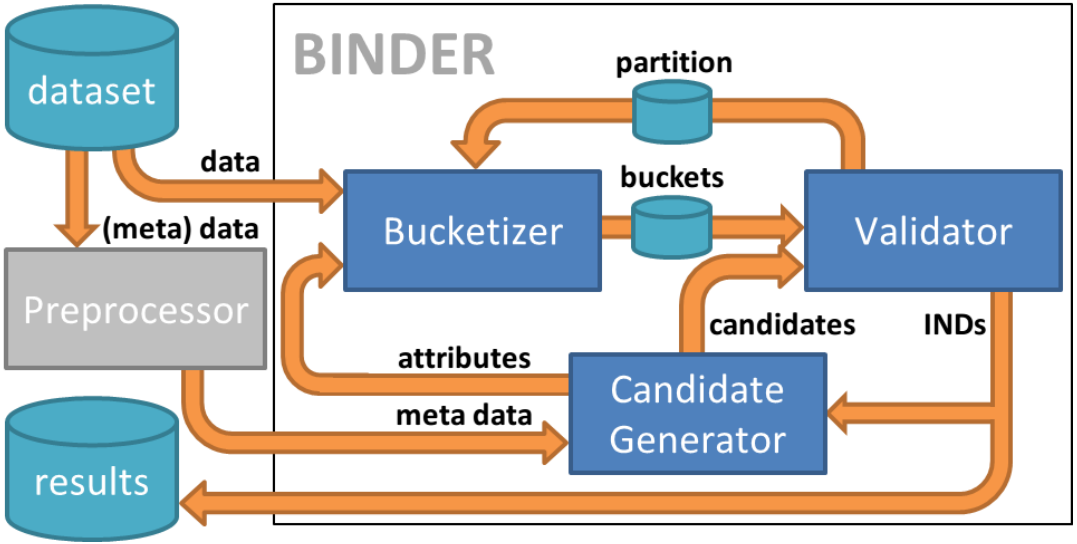
Motivation

Distributed Computing Challenges



Questions

? How do we change a particular algorithm to efficiently execute on various, independent and heterogeneous computing elements?



hadoop Map Reduce

APACHE STORM™

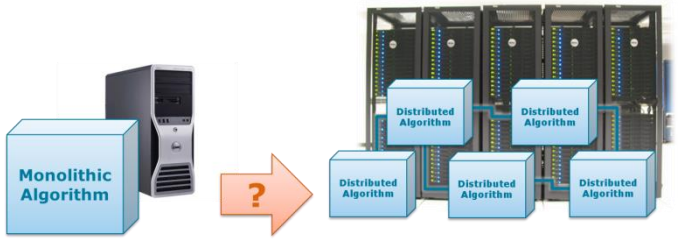
APACHE Spark™

Apache Flink

TEZ

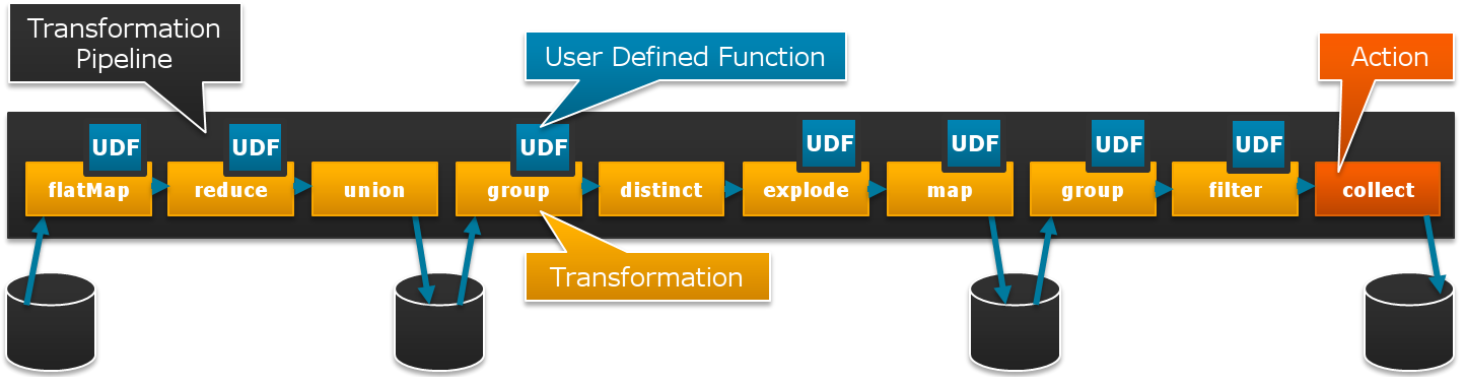
Motivation

Distributed Computing Challenges



Questions

? How do we change a particular algorithm to efficiently execute on various, independent and heterogeneous computing elements?



hadoop Map Reduce

APACHE STORM™

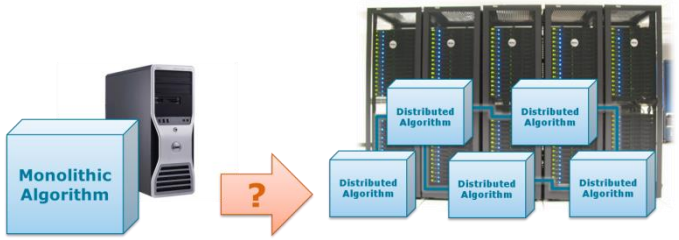
APACHE Spark™

Apache Flink

TEZ

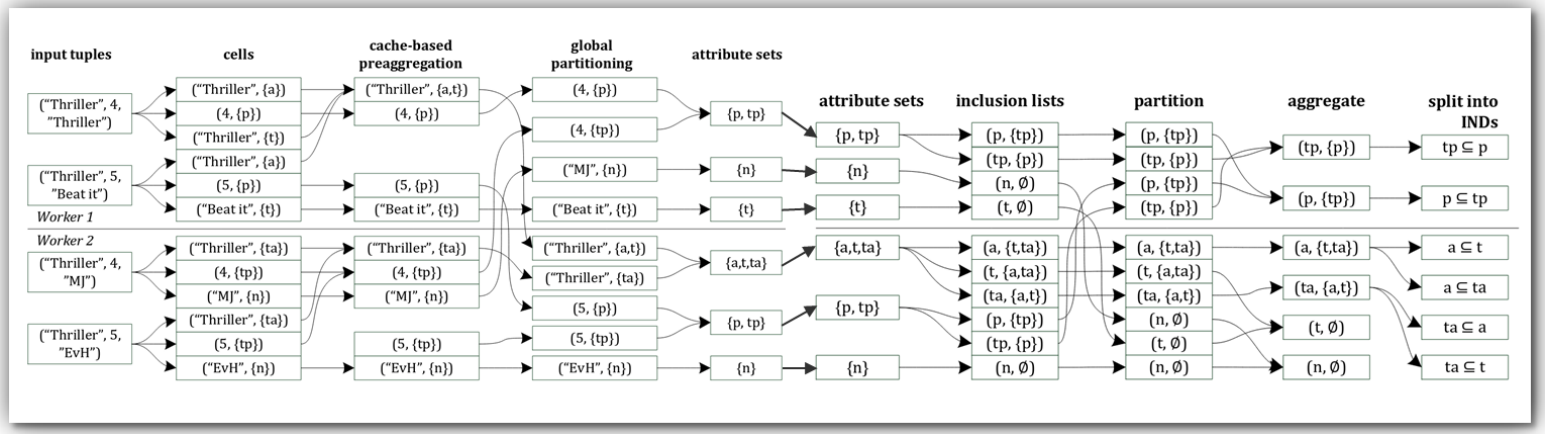
Motivation

Distributed Computing Challenges



Questions

? How do we change a particular algorithm to efficiently execute on various, independent and heterogeneous computing elements?



hadoop MapReduce

APACHE STORM™

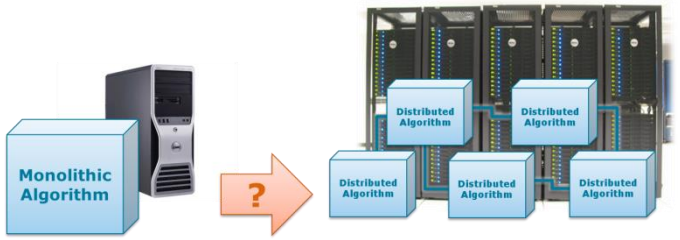
APACHE Spark™

Apache Flink

TEZ

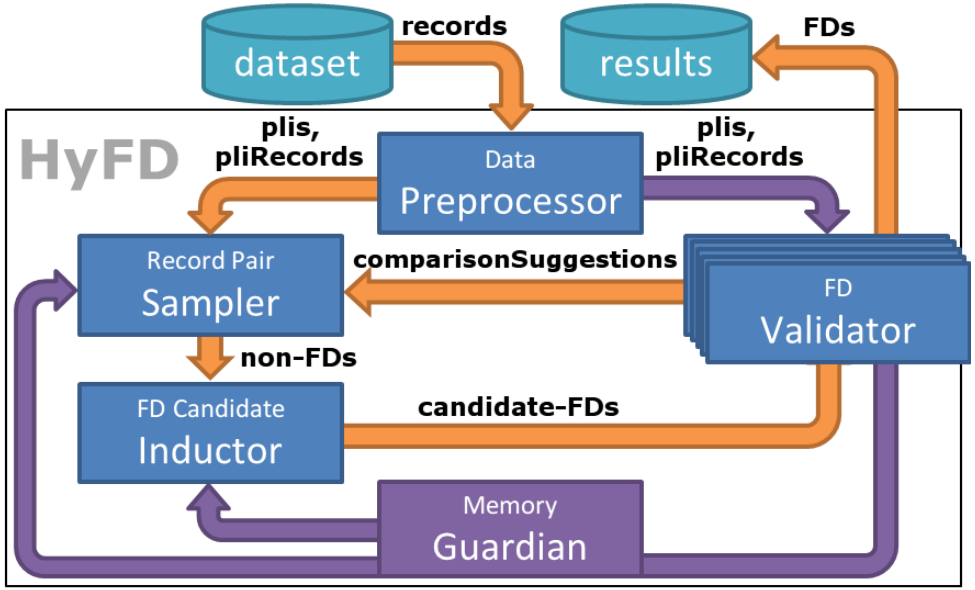
Motivation

Distributed Computing Challenges



Questions

? How do we change a particular algorithm to efficiently execute on various, independent and heterogeneous computing elements?



hadoop MapReduce

APACHE STORM™

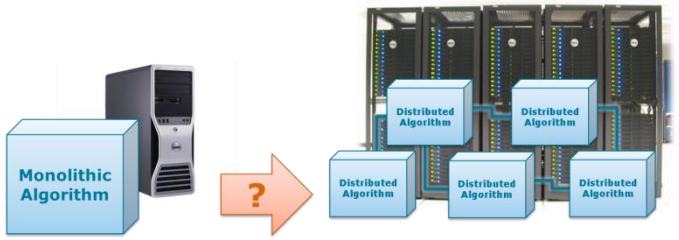
APACHE Spark™

Apache Flink

TEZ

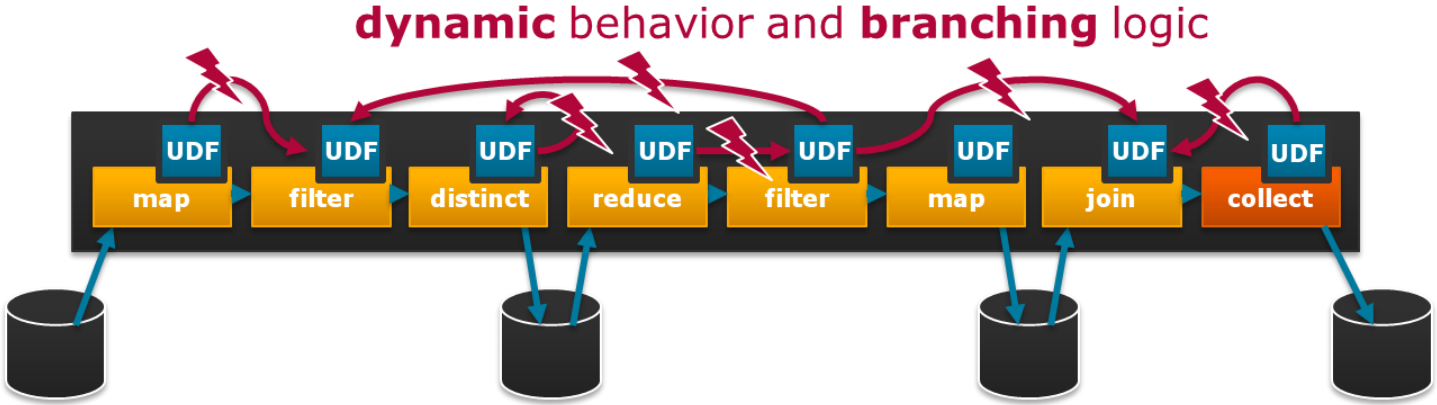
Motivation

Distributed Computing Challenges



Questions

? How do we change a particular algorithm to efficiently execute on various, independent and heterogeneous computing elements?



hadoop Map Reduce

APACHE STORM™

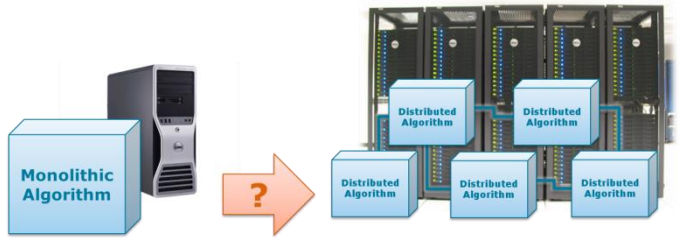
APACHE Spark™

Apache Flink

TEZ

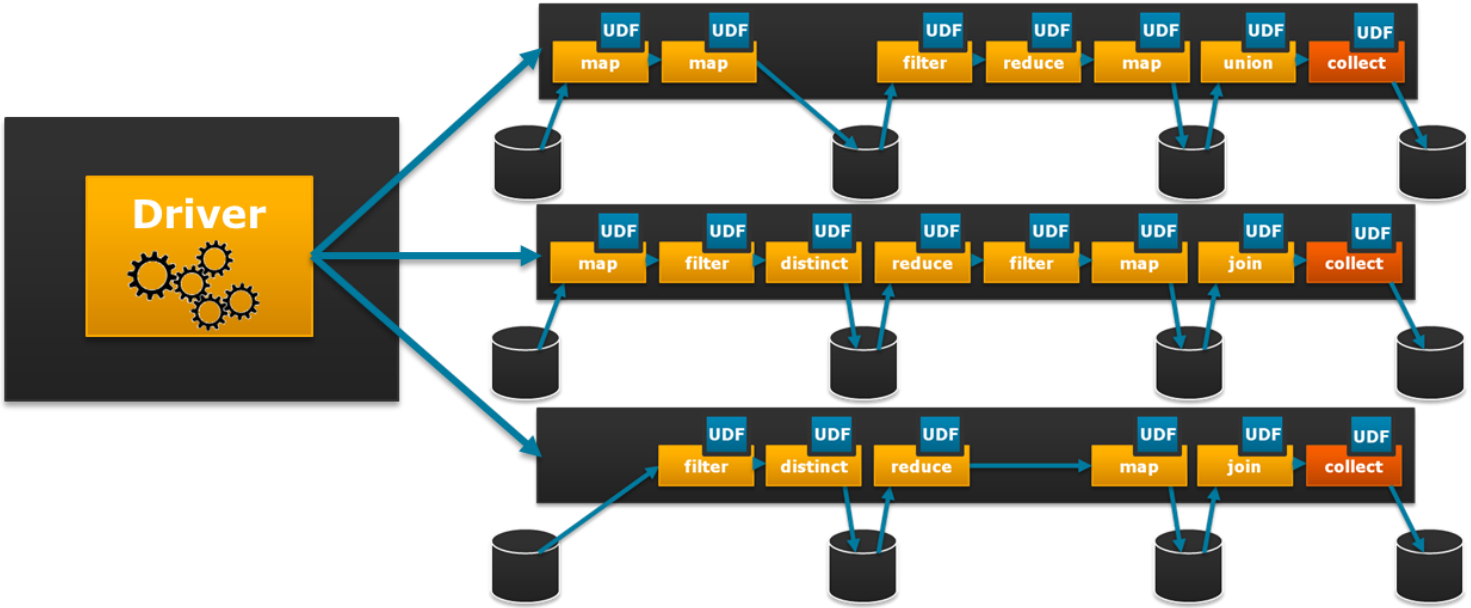
Motivation

Distributed Computing Challenges



Questions

? How do we change a particular algorithm to efficiently execute on various, independent and heterogeneous computing elements?



hadoop Map Reduce

APACHE STORM™

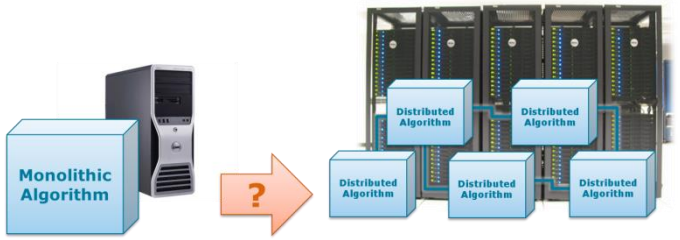
APACHE Spark™

Apache Flink

TEZ

Motivation

Distributed Computing Challenges



Questions

- ✗ How do we change a particular algorithm to efficiently execute on various, independent and heterogeneous computing elements?
- ✓ How do we utilize all available resources in an optimal way?
- ✓ How does the algorithm deal with the increased error susceptibility of a parallel/distributed system?
- ✓ How can the algorithm support elasticity, i.e., sets of computing resources that change at runtime?
- ✓ How does a parallel/distributed system control its resource consumption in terms of overall memory and CPU usage?
- ✓ How do we start and terminate such systems?
- ✓ How do we debug, monitor, and profile them?

hadoop MapReduce

APACHE STORM™

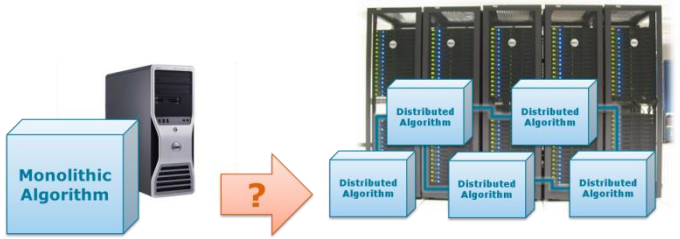
APACHE Spark™

Apache Flink

TEZ

Motivation

Distributed Computing Challenges



Questions

- How do we change a particular algorithm to efficiently execute on various, independent and heterogeneous computing elements?
- How do we utilize all available resources in an optimal way?
- How does the algorithm deal with the increased error susceptibility of a parallel/distributed system?
- How can the algorithm support elasticity, i.e., sets of computing resources that change at runtime?
- How does a parallel/distributed system control its resource consumption in terms of overall memory and CPU usage?
- How do we start and terminate such systems?
- How do we debug, monitor, and profile them?

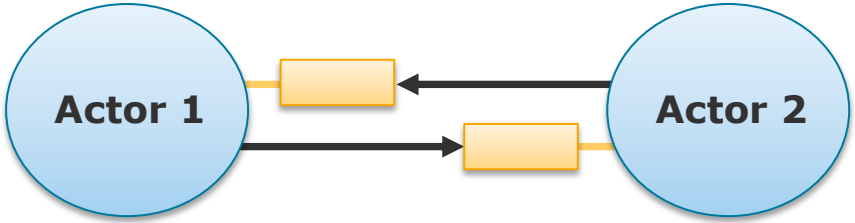
akka

Orleans

Actor Programming

Object-oriented programming	Actor programming	Task-oriented programming
▪ Objects encapsulate state and behavior ▪ Objects communicate with each other	▪ Actors encapsulate state and behavior ▪ Actors communicate with each other ▪ Actor activities are scheduled and executed transparently	▪ Application split down into task graph ▪ Tasks are scheduled and executed transparently
➢ Separation of concerns makes applications easier to build and maintain.	➢ Combines the advantages of object- and task-oriented programming.	➢ Decoupling of tasks and resources allows for asynchronous and parallel programming.

Actor Model

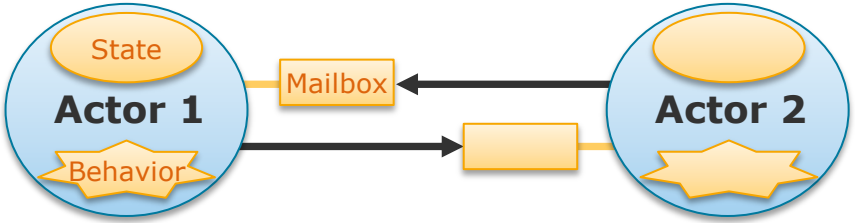


Actor Model

- A stricter message-passing model that treats actors as the universal primitives of concurrent computation
- Actor:
  - Computational entity (private state/behavior)
  - Owns exactly one mailbox
  - Reacts on messages it receives (one message at a time)
- Actor reactions:
  - Send a finite number of messages to other actors
  - Create a finite number of new actors
  - Change own state, i.e., behavior for next message
- Actor model prevents many parallel programming issues (race conditions, locking, deadlocks, …)

"The actor model retained more of what I thought were good features of the object idea"
Alan Kay, pioneer of object orientation

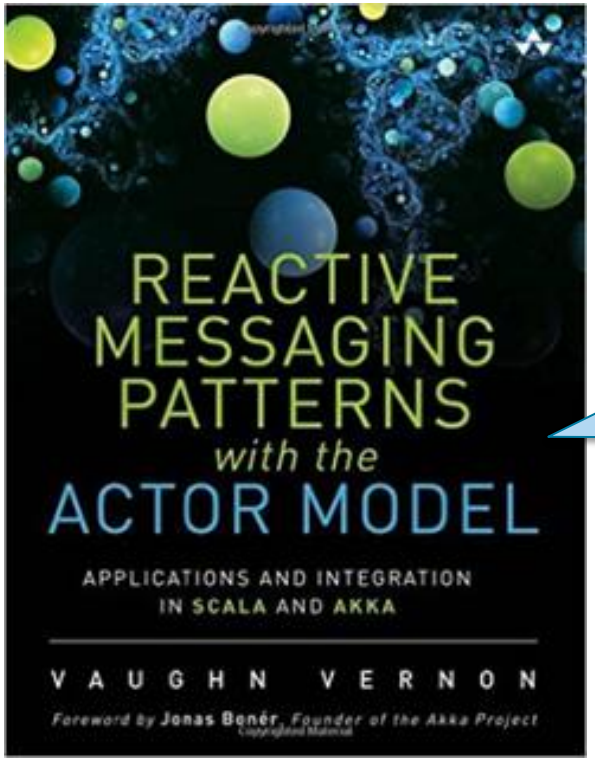
Actors



- Actor = State + Behavior + Mailbox
- Communication:
  - Sending messages to mailboxes
  - Unblocking, fire-and-forget
- Messages:
  - Immutable, serializable objects
  - Object classes are known to both sender and receiver
  - Receiver interprets a message via pattern matching

Mutable messages are possible, **but don't use them!**

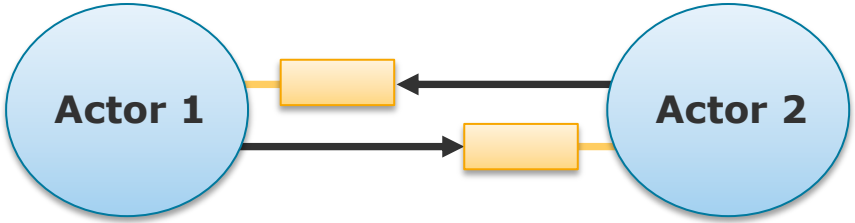
Patterns



Actor programming is a mathematical model that defines basic rules for communication
(not a style guide for architecture)

Writing actor-based systems is based on patterns

Frameworks



Popular Actor Frameworks

- Erlang:
  - Actor framework already included in the language
  - First popular actor implementation
  - Most consistent actor implementation
    (best native support and strongest actor isolation)
- Akka:
  - Actor framework for the JVM (Java and Scala)
  - Most popular actor implementation (at the moment)
- Orleans:
  - Actor framework for Microsoft .NET

Toolkit and Runtime

- A free and open-source toolkit and runtime for building concurrent and distributed applications on the JVM
- Supports multiple programming models for concurrency, but emphasizes actor-based concurrency
- Inspired by Erlang
- Written in Scala (included in the Scala standard library)
- Offers interfaces for Java and Scala

Akka Modules

Akka Actors

Core Actor model classes for concurrency and distribution

Akka Cluster

Classes for the resilient and elastic distribution over multiple nodes

Akka Streams

Asynchronous, non-blocking, backpressured, reactive stream classes

Akka Http

Asynchronous, streaming-first HTTP server and client classes

Cluster Sharding

Classes to decouple actors from their locations referencing them by identity

Akka Persistence

Classes to persist actor state for fault tolerance and state restore after restarts

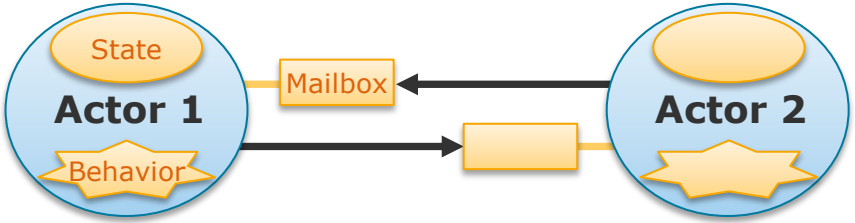
Distributed Data

Classes for an eventually consistent, distributed, replicated key-value store

Alpakka

Stream connector classes to other technologies

Akka Actors



Called in default actor constructor and set as the actor's behavior

```
public class Wor  extends AbstractActor {
  @Override
  public Receive createReceive() {
    return receiveBuilder()
      .match(String.class, this::respondTo)
      .matchAny(object -> System.out.println("Could not understand received message"))
      .build();
  }
  private void respondTo(String message) {
    System.out.println(message);
    this.sender().tell("Received your message, thank you!", this.self());
  }
}
```

Inherit default actor behavior, state and mailbox implementation

The **Receive** class performs pattern matching and de-serialization

A builder pattern for constructing a **Receive** object with otherwise many constructor arguments

Send a response to the sender of the last message (asynchronously, non-blocking)

Actor Hierarchies

Task- and data-parallelism

- Actors can dynamically create new actors

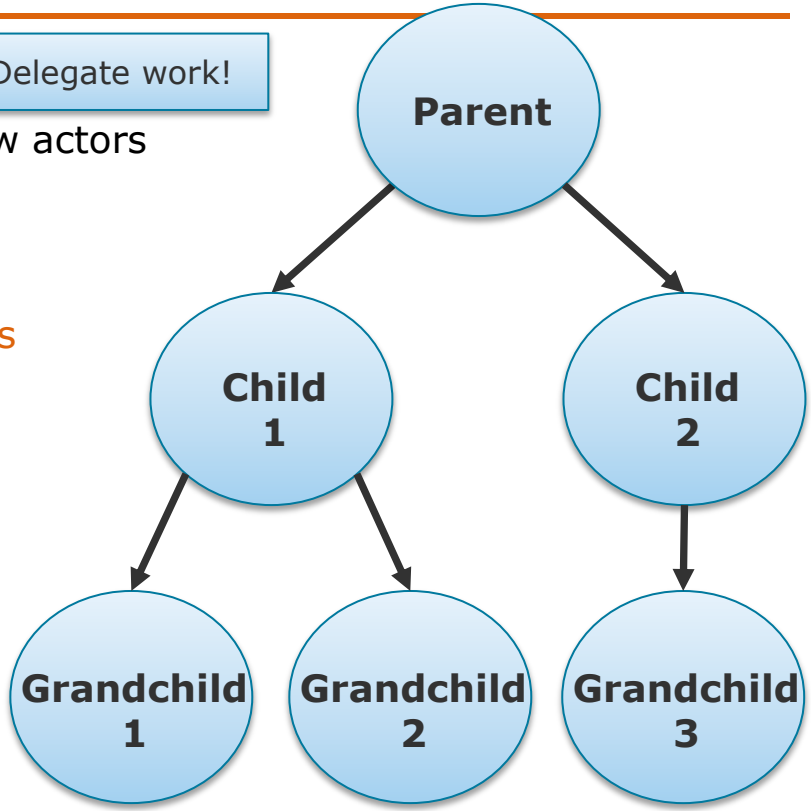
Delegate work!

Supervision hierarchy

- Creating actor (parent) supervises created actor (child)

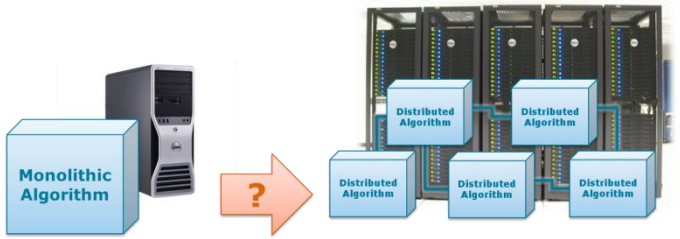
Fault-tolerance

- If child fails, parent can choose:
  - restart, resume, stop, or escalate



Motivation

Distributed Computing Challenges



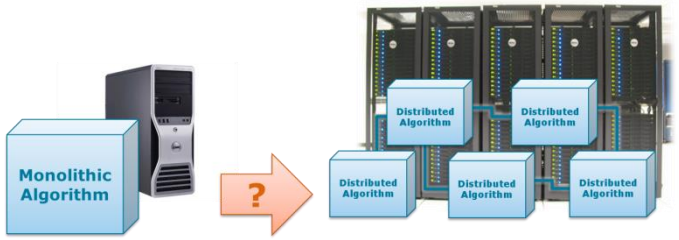
Questions

- ? How do we change a particular algorithm to efficiently execute on various, independent and heterogeneous computing elements?
- ? How do we utilize all available resources in an optimal way?
- ? How does the algorithm deal with the increased error susceptibility of a parallel/distributed system?
- ? How can the algorithm support elasticity, i.e., sets of computing resources that change at runtime?
- ? How does a parallel/distributed system control its resource consumption in terms of overall memory and CPU usage?
- ? How do we start and terminate such systems?
- ? How do we debug, monitor, and profile them?

akka

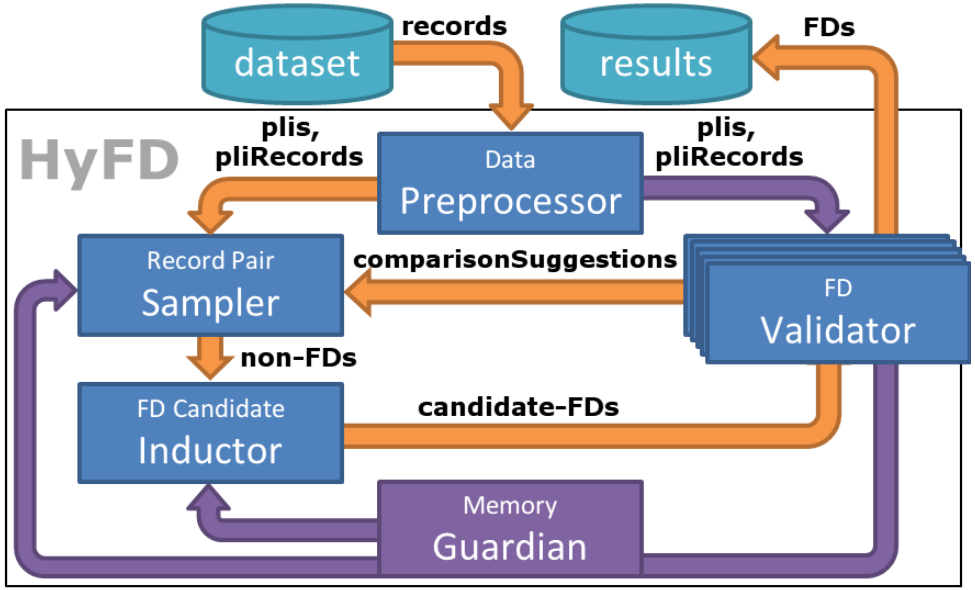
Orleans

Distributed Computing Challenges

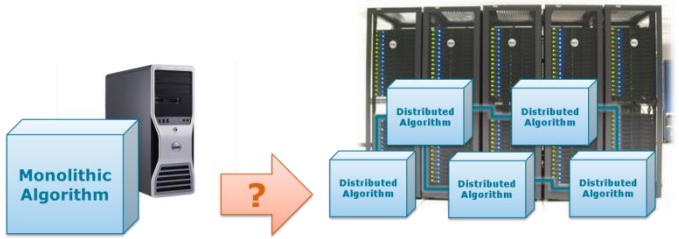


Questions

? How do we change a particular algorithm to efficiently execute on various, independent and heterogeneous computing elements?



Distributed Computing Challenges

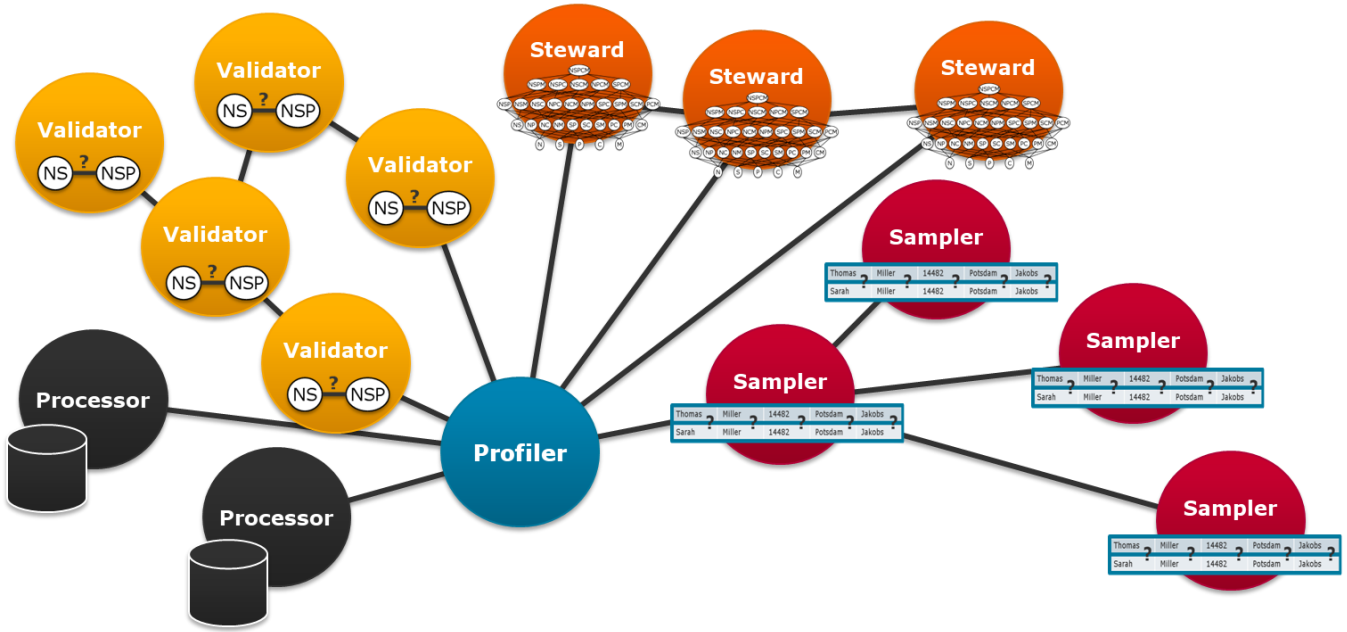


Questions

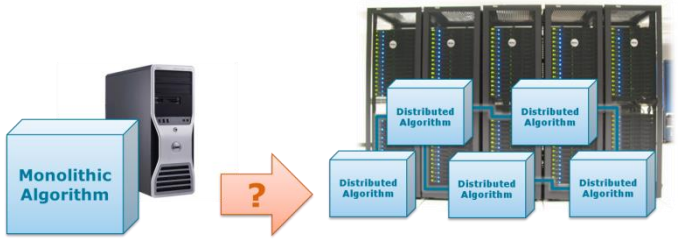
? How do we change a particular algorithm to efficiently execute on various, independent and heterogeneous computing elements?

akka

Orleans



Seminar



Learning Goals

a) Learn state-of-the-art distributed computing techniques.
b) Build a large (and complex) distributed algorithm.
c) Solve problems that arise from distributed computing.
d) Approach an algorithmic challenge in a scientific way.
e) Reveal new research questions for distributed computing.

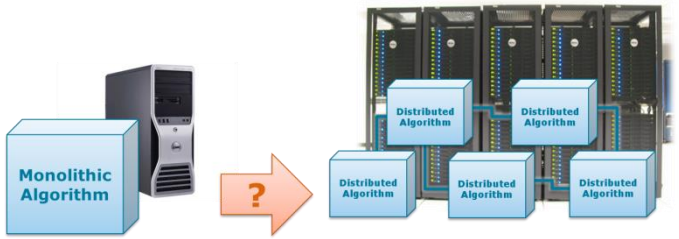
akka

Orleans

Project Tasks

1) Choose a topic.
2) Study the literature of your topic (books, papers, and online material).
3) Design a distributed algorithm that solves the problem of your topic.
4) Evaluate your solution w.r.t. performance, scalability, and robustness.
5) Document your approach by writing a scientific paper about it.

Your Distributed Algorithm



… optimizes resource utilization:
All nodes contribute to the overall task. Situations where cluster nodes are idle while others are not should be avoided as far as possibly.
(work scheduling, load balancing, cluster metrics, actor migration, ...)

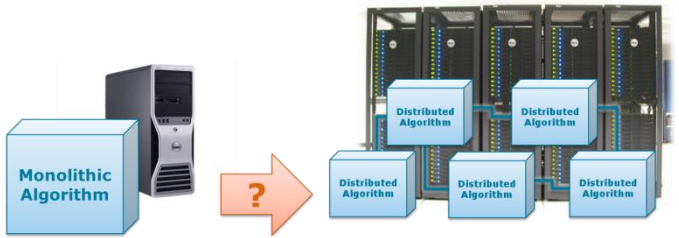
… tolerates node failures:
Failing nodes do not cause the computation to fail. Their subtasks are re-scheduled and their states are recovered, if needed.
(reliable message sending, master-worker pattern, consensus techniques, ...)

… enables cluster growth:
Nodes are able to enter the cluster and your computation. They are dynamically provided with work.
(dynamic membership management, work stealing, work scheduling, ...)

… resolve memory overflows:
Memory issues are dynamically detected and resolved. Applying drastical measures to avoid OOM situations is OK.
(disk spilling, result pruning, dynamic compression, ...)

… starts and stops in a clean way:
The distributed processing starts easily. When it ends, all nodes shutdown cleanly and the result is stored safely.
(scripted startup, reaper pattern, coordinated shutdown, ...)

Topic Suggestions



(1) Distributed Order Dependency Discovery

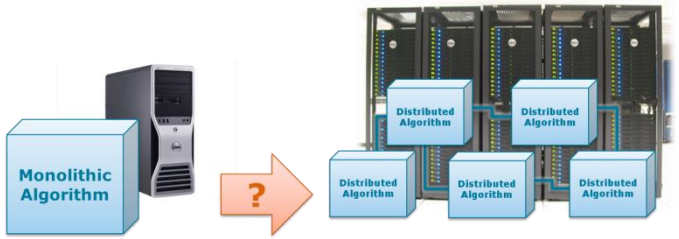
- An order dependency is a statement of the form $X \rightarrow Y$ over a schema R specifying that ordering a relational instance r of R by the attribute list $X \subset R$ also orders r by the attribute list $Y \subset R$, i.e., $X \rightarrow Y$ is true in r, iff $\forall s, t \in r$: $s[X] \leq t[X] \Rightarrow s[Y] \leq t[Y]$.
  - In other words, the order of X determines the order of Y.
- Example:

  $Weight \rightarrow Size$

  $ID \not\rightarrow Size$

ID	Name	Weight	Size	Type	Weak	Strong
25	Pikachu	6.0	0.4	electric	ground	water
29	Nidoran	9.0	0.5	poison	ground	gras
37	Vulpix	9.9	0.6	fire	water	ice
26	Raichu	12.0	0.8	electric	ground	water
63	Abra	19.5	0.9	psychic	ghost	fighting
38	Ninetails	19.9	1.1	fire	water	ice
65	Alakazam	54.3	1.5	psychic	ghost	fighting
64	Kadabra	56.5	1.6	psychic	ghost	fighting
150	Mewtwo	122.0	2.0	psychic	ghost	fighting

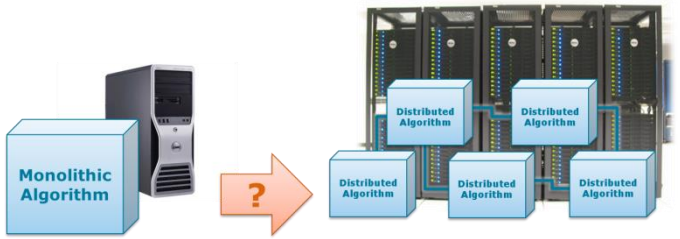
Topic Suggestions



(1) Distributed Order Dependency Discovery

- An order dependency is a statement of the form $X \rightarrow Y$ over a schema R specifying that ordering a relational instance r of R by the attribute list $X \subset R$ also orders r by the attribute list $Y \subset R$, i.e., $X \rightarrow Y$ is true in r, iff $\forall s, t \in r$: $s[X] \leq t[X] \Rightarrow s[Y] \leq t[Y]$.
- Task:
  - Given a relational instance r, find all order dependencies.
- Literature:
  - "Order Dependency in the Relational Model",
    Seymour Ginsburg and Richard Hull, Theoretical Computer Science, 1983
  - "Fundamentals of Order Dependencies",
    Jaroslaw Szlichta et al., PVLDB, 2012
  - "Effective and Complete Discovery of Order Dependencies via Set-based Axiomatization",
    Jaroslaw Szlichta et al., PVLDB, 2017
  - "Discovering Order Dependenciesthrough Order Compatibility",
    Cristian Consonni et al., EDBT, 2019

Topic Suggestions



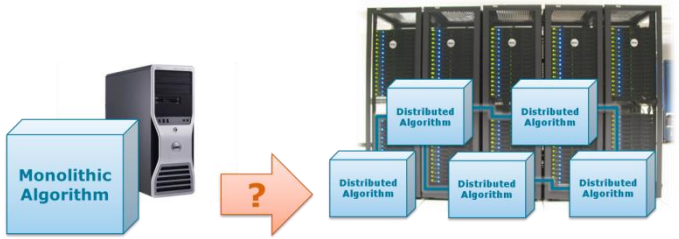
(2) Distributed Entity Resolution

- A duplicate (s,t) is a pair of two records s and t that describe the same real-world entity. Two records are considered to be a duplicate, iff their similarity sim(s,t) w.r.t. a given similarity function sim() is larger than a specified threshold x, i.e., iff sim(s,t) > x.
- Example:

(25,26)

ID	Name	Weight	Size	Type	Weak	Strong
25	pikachu	6.0	0.4	electric	ground	water
29	Nidoran	9.0	0.5	poison	ground	gras
26	Pikachu	6.0	0.4	Electric	Ground	Water
63	Abra	19.5	0.9	psychic	ghost	fighting
38	Ninetails	19.9	1.1	fire	water	ice
65	Alakazam	54.3	1.5	psychic	ghost	fighting
64	Kadabra	56.5	1.3	psychic	ghost	fighting
150	Mewtwo	122.0	2.0	psychic	ghost	fighting

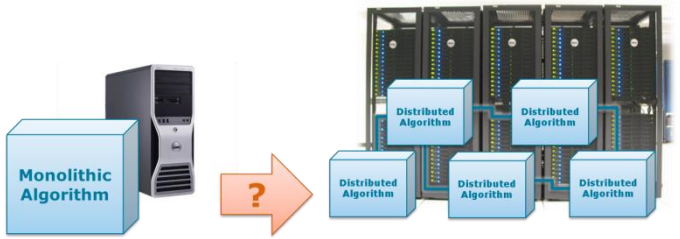
Topic Suggestions



(2) Distributed Entity Resolution

- A duplicate (s,t) is a pair of two records s and t that describe the same real-world entity. Two records are considered to be a duplicate, iff their similarity sim(s,t) w.r.t. a given similarity function sim() is larger than a specified threshold x, i.e., iff sim(s,t) > x.
- Task:
  - Given a relational instance r, find all real-world entities.
- Literature:
  - "An Introduction to Duplicate Detection",
    Felix Naumann and Melanie Herschel, Synthesis Lecture on Data Management, 2010
  - "Data Matching - Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection",
    Peter Christen, Springer, 2012
  - "Scalable Duplicate Detection utilizing Apache Spark",
    Niklas Wilcke, Master Thesis, 2015
    https://vsis-www.informatik.uni-hamburg.de/getDoc.php/thesis/811/scalable-duplicate-detection-2015-niklas-wilcke.pdf

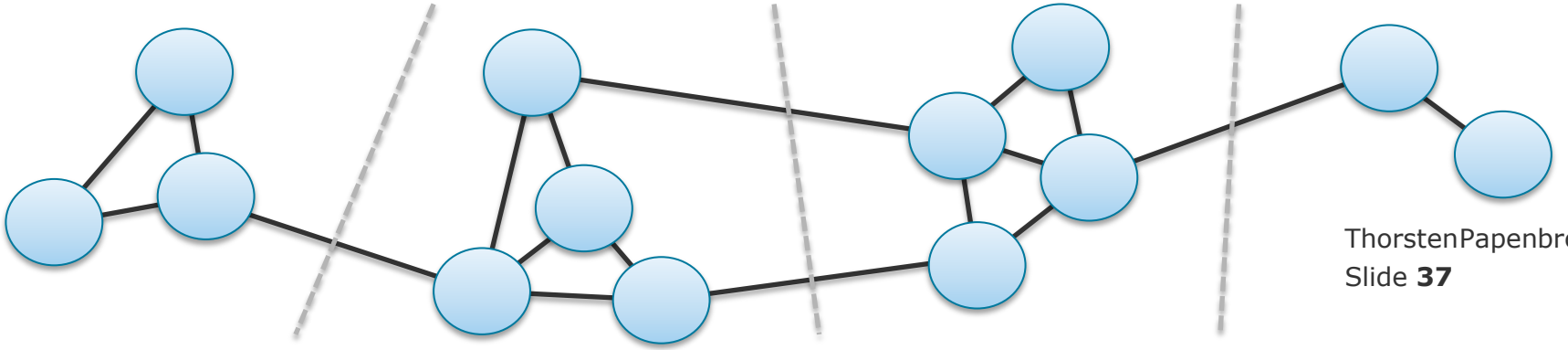
Topic Suggestions



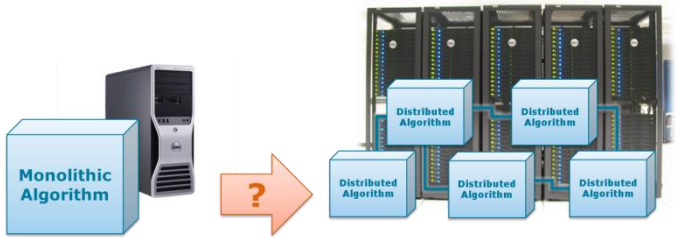
(3) Distributed Graph Queries

Given: A graph G = (E,V) with edges E and vertices V.

- An all shortest paths query calculates the shortest path between all pairs of vertices (V_1,V_2).
- An all maximum cliques query calculated all sets of vertices $V' \subset V$ such that all vertices in V' are connected by an edge (i.e., $\forall V_1,V_2 \in V'$: $(V_1,V_2) \in E$) and V' is maximal.
- A graph clustering with minimal cutting edges query divides a graph into k clusters such that the clusters are of similar size and the number of edges connection clusters is minimal.



Topic Suggestions

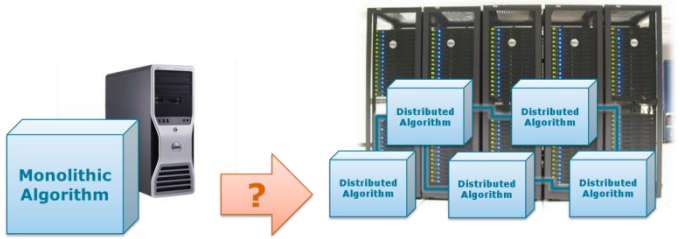


(3) Distributed Graph Queries

Given: A graph G = (E,V) with edges E and vertices V.

- An all shortest paths query calculates the shortest path between all pairs of vertices (V_1,V_2).
- An all maximum cliques query calculated all sets of vertices $V' \subset V$ such that all vertices in V' are connected by an edge (i.e., $\forall V_1,V_2 \in V'$: $(V_1,V_2) \in E$) and V' is maximal.
- A graph clustering with minimal cutting edges query divides a graph into k clusters such that the clusters are of similar size and the number of edges connection clusters is minimal.
- Task:
  - Given a graph G, answer the query
    (and repartition the graph based on the result with minimal data copy effort).
- Literature:
  - "System G Distributed Graph Database",
    Gabriel Tanase et al., https://arxiv.org/pdf/1802.03057.pdf, 2018 (see related work)

Topic Suggestions



(4) Distributed Data Integration

- A data integration process takes two (or more) schemata R_1 and R_2 with their relational instances r_1 and r_2 and outputs one schema R and its relational instance r that holds all the data from r_1 and r_2. The process uses the data (e.g. r_1 and r_2) and metadata (e.g. data types, statistics, and dependencies of r_1 and r_2) to find matching relations and attributes.

Name	Type	Equatorial diameter	Mass	Orbital radius	Orbital period	Rotation period	Confirmed moons	Rings	Atmosphere
Mercury	Terrestrial	0.382	0.06	0.47	0.24	58.64	0	no	minimal
Venus	Terrestrial	0.949	0.82	0.72	0.62	−243.02	0	no	CO_2, N_2
Earth	Terrestrial	1.000	1.00	1.00	1.00	1.00	1	no	N_2, O_2, Ar
Mars	Terrestrial	0.532	0.11	1.52	1.88	1.03	2	no	CO_2, N_2, Ar
Jupiter	Giant	11.209	317.8	5.20	11.86	0.41	67	yes	H_2, He
Saturn	Giant	9.449	95.2	9.54	29.46	0.43	62	yes	H_2, He
Uranus	Giant	4.007	14.6	19.22	84.01	−0.72	27	yes	H_2, He
Neptune	Giant	3.883	17.2	30.06	164.8	0.67	14	yes	H_2, He

Planet	Rotation Period	Revolution Period
Mercury	58.6 days	87.97 days
Venus	243 days	224.7 days
Earth	0.99 days	365.26 days
Mars	1.03 days	1.88 years
Jupiter	0.41 days	11.86 years
Saturn	0.45 days	29.46 years
Uranus	0.72 days	84.01 years
Neptune	0.67 days	164.79 years
Pluto	6.39 days	248.59 years

Symbol	Unicode	Glyph
Sun	U+2609	☉
Moon	U+263D	☽
Moon	U+263E	☾
Mercury	U+263F	☿
Venus	U+2640	♀
Earth	U+1F728	🜨
Mars	U+2642	♂
Jupiter	U+2643	♃
Saturn	U+2644	♄
Uranus	U+2645	♅
Uranus	U+26E2	⛢
Neptune	U+2646	♆
Eris	≈ U+2641	♁
Eris	≈ U+29EC	⧬
Pluto	U+2647	♇
Pluto	not present	--
Aries	U+2648	♈
Taurus	U+2649	♉
Gemini	U+264A	♊
Cancer	U+2648	♋
Leo	U+264C	♌
Virgo	U+264D	♍
Libra	U+264E	♎
Scorpio	U+264F	♏
Sagittarius	U+2650	♐
Capricorn	U+2651	♑
Capricorn	U+2651	♑
Aquarius	U+2652	♒
Pisces	U+2653	♓
Conjunction	U+260C	☌
...	...	...

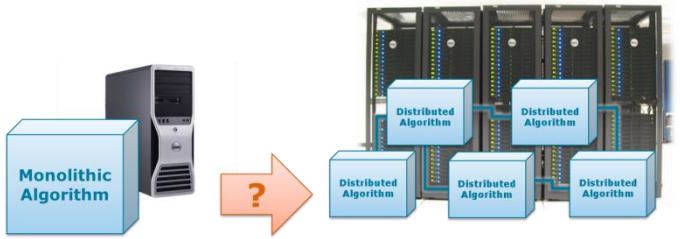
Planet	Synodic period	Synodic period (mean)	Days in retrograde
Mercury	116	3.8	~21
Venus	584	19.2	41
Mars	780	25.6	72
Jupiter	399	13.1	121
Saturn	378	12.4	138
Uranus	370	12.15	151
Neptune	367	12.07	158

Planet	Mean distance	Relative mean distance
Mercury	57.91	1
Venus	108.21	1.86859
Earth	149.6	1.3825
Mars	227.92	1.52353
Ceres	413.79	1.81552
Jupiter	778.57	1.88154
Saturn	1,433.53	1.84123
Uranus	2,872.46	2.00377
Neptune	4,495.06	1.56488
Pluto	5,869.66	1.3058

Sign	House	Domicile	Detriment	Exaltation	Fall	Planetary Joy
Aries	1st House	Mars	Venus	Sun	Saturn	Mercury
Taurus	2nd House	Venus	Pluto	Moon	Uranus	Jupiter
Gemini	3rd House	Mercury	Jupiter	N/A	N/A	Saturn
Cancer	4th House	Moon	Saturn	Jupiter	Mars	Venus
Leo	5th House	Sun	Uranus	Neptune	Mercury	Mars
Virgo	6th House	Mercury	Neptune	Pluto, Mercury	Venus	Saturn
Libra	7th House	Venus	Mars	Saturn	Sun	Moon
Scorpio	8th House	Pluto	Venus	Uranus	Moon	Saturn
Sagittarius	9th House	Jupiter	Mercury	N/A	N/A	Sun
Capricorn	10th House	Saturn	Moon	Mars	Jupiter	Mercury
Aquarius	11th House	Uranus	Sun	Mercury	Neptune	Venus
Pisces	12th House	Neptune	Mercury	Venus	Pluto, Mercury	Moon

Planet	Calculated (in AU)	Observed (in AU)	Perfect octaves	Actual distance
Mercury	0.4	0.387	0	0
Venus	0.7	0.723	1	1.1
Earth	1	1	2	2
Mars	1.6	1.524	4	3.7
Asteroid belt	2.8	2.767	8	7.8
Jupiter	5.2	5.203	16	15.7
Saturn	10	9.539	32	29.9
Uranus	19.6	19.191	64	61.4
Neptune	38.8	30.061	96	-96.8
Pluto	77.2	39.529	128	127.7

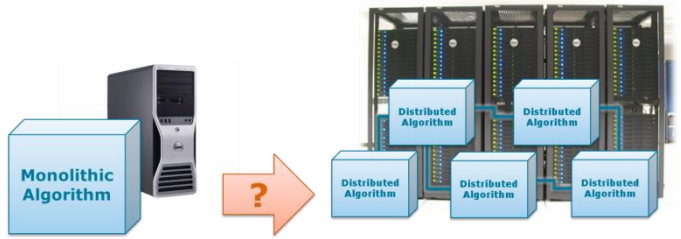
Topic Suggestions



(4) Distributed Data Integration

- A data integration process takes two (or more) schemata R_1 and R_2 with their relational instances r_1 and r_2 and outputs one schema R and its relational instance r that holds all the data from r_1 and r_2. The process uses the data (e.g. r_1 and r_2) and metadata (e.g. data types, statistics, and dependencies of r_1 and r_2) to find matching relations and attributes.
- Task:
  - Given some input schemata, find all attribute correspondences, then derive an output schema and populate the output schema with the input data.
- Literature:
  - "Principles of Data Integration",
    AnHai Doan, Alon Halevy, and Zachary Ives, Elsevier, 2012
  - "The Clio project:Managing heterogeneity",
    R. J. Miller et al., SIGMOD Record, 2001

Topic Suggestions



(5) Distributed Index Maintenance

(6) Distributed Data Mining

(7) Distributed Block Chain Algorithms

(8) Distributed Model Training

Reliable Distributed Systems Engineering

Dataset Suggestions

Enigma
https://public.enigma.com/

Data.Gov
https://www.data.gov/

Web Data Commons – Web Table Corpora
http://webdatacommons.org/webtables/index.html

MusicBrainz @BitBucket
https://bitbucket.org/metabrainz/musicbrainz-server

The GDELT Project
https://www.gdeltproject.org/

UCI Machine Learning Repository
https://archive.ics.uci.edu/ml/index.php

Kaggle
https://www.kaggle.com/competitions

Awesome Data @GitHub
https://github.com/awesomedata/awesome-public-datasets

Quora
https://www.quora.com/Data/Where-can-I-find-large-datasets-open-to-the-public

Organization

Tasks

1) Choose a topic.
2) Study the literature of your topic (books, papers, and online material).
3) Design a distributed algorithm that solves the problem of your topic.
4) Evaluate your solution w.r.t. performance, scalability, and robustness.
5) Document your approach by writing a scientific paper about it.

Grading

- 10% Active participation at all meetings
- 20% Implementation of a distributed algorithm for your team-specific challenge
- 10% Evaluation of your distributed algorithm
- 40% A midterm and a final presentation
- 20% A written summary/documentation of your distributed algorithm
  - 2-4 pages per person according to ACM template

Organization

Prerequisites

- Database knowledge (ideally Database System I and Database Systems II)
- Actor programming knowledge (ideally Distributed Data Analytics or Distributed Data Management)

Metadata

- Extent: 4 SWS
- Location: Campus II (F-E.06 for large meetings and F-2.04 for small meetings)
- Dates: Tuesdays, 09:15 - 10:45 AM (for large meetings)
- Class: At most 8 participants (4 teams á 2 students)
- Register: Informal email to thorsten.papenbrock@hpi.de by April 12 (notification April 15)

Registration Email

- Add your actor programming experience (e.g. DDA, DDM, some other course, or project).
- Add a ranking of up to three topics that interest you (from the list shown today or own suggestions).
  - We do the final topic assignment in our first meeting; so this is not a commit!
- <optional> Add a team partner; you get either accepted or rejected together if seats get tight.

Organization

Small team meetings

- An early meeting to define the topic details
- A meeting before each presentation
- A meeting to discuss the paper write-up
- On request, whenever something needs to be discussed

Schedule (tentative)

- April 12: (Email) Registration
- April 15: (Email) Notification
- April 16: (Meeting) Team building and topic selection
- April 16-30: (Small meetings) Topic definitions
- April 30: (Meeting) Informal topic and idea presentations/discussions

Project duration

- We set a deadline in consensus with all teams once the registration period is over.