

RUPRECHT-KARLS-
UNIVERSITÄT
HEIDELBERG



Accelerating scientific computing without driving up the costs

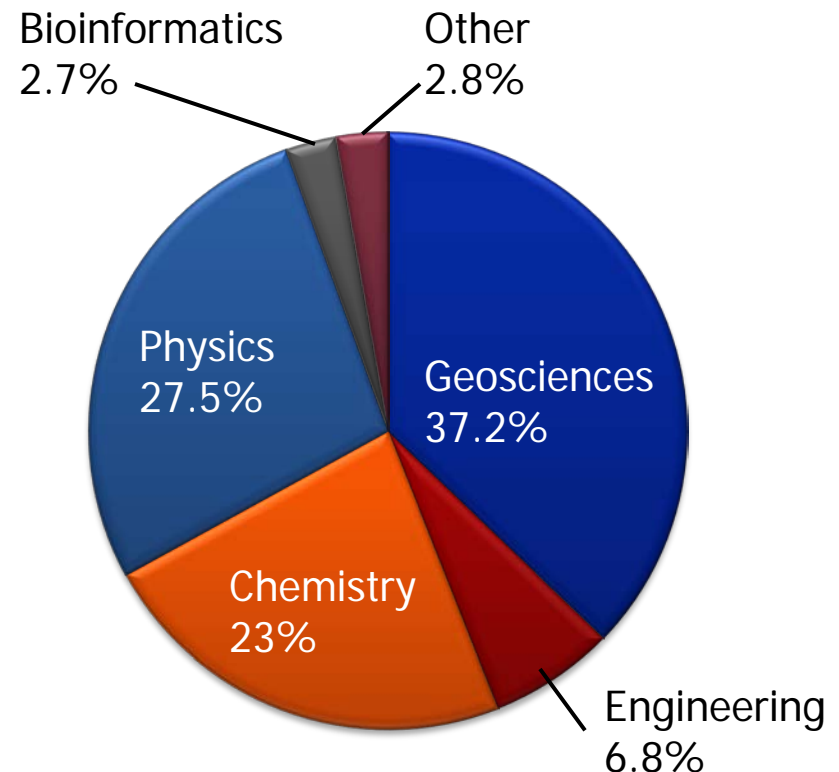
Prof. Dr. Artur Andrzejak

<http://pvs.ifi.uni-heidelberg.de/>

artur@uni-hd.de

High-Performance Computing at ZIB Berlin

- ZIB operates a High Performance Computing (HPC) system **HLRN-II**
- Ranked 39 in Top500 Supercomputing Sites
- Researchers (users) want to compute faster – why?
- Usage of HLRN-I in 2008 (previous system)



Time-to-Solution is Important

- Researchers work to a large extent in an "exploratory" way
 - They evaluate last results and decide *then* on the next job(s)
 - Waiting time for a batch computation to finish is called **Time-to-Solution**
- Shortening the **Time-to-Solution** reduces significantly the total time of exploration



Faster Computing => More Processors?

- Parallelization is the primary way to speed up computations
- **Processor count in supercomputers increases** each year
 - Speed of a rank X supercomputer doubles every 13 months
 - But # transistors ~ # cores per chip doubles only every 24 months
- This implies **higher costs - mainly due to power**

	Cray X-MP/24	Cray Y-MP4/264	Cray T3D SC 192	Cray T3E	IBM 690 pSeries	SGI ICE
Year	1986	1992	1994	1997	2002	2008
GFlops/s	0.47	1.33	38	363	2662	125,000
# Processors	2	4	192	512	512	~ 2500
Power (kW)				90	160	600

Supercomputers at ZIB

Making Time-To-Solution **Cheaper**

A. Online Computing with MapReduce

Using preliminary results for accelerating decisions about next exploratory steps

B. Resources with Heterogeneous Availability

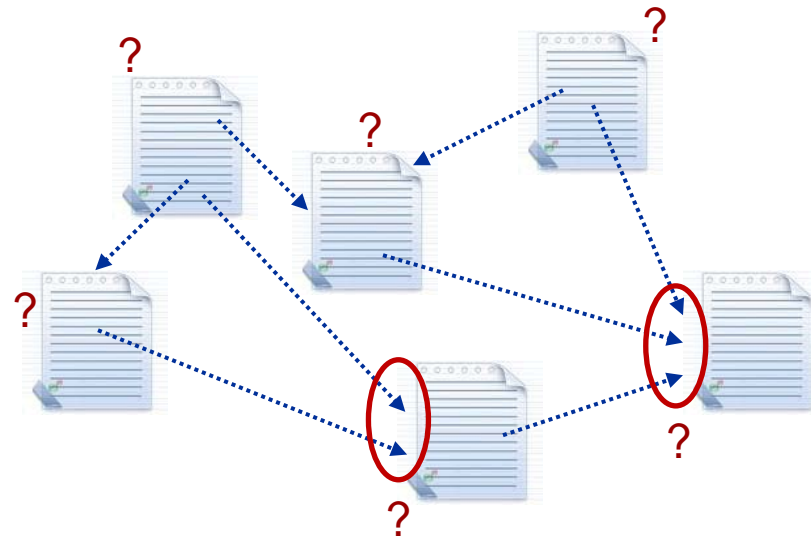
Combining highly available (and thus expensive) resources with less available but cheaper ones

ONLINE COMPUTING WITH MAPREDUCE

MapReduce Programming Model

- Re-discovered by Google with goals:
 - "Reliability has to come from the software"
 - "How can we make it easy to write distributed programs?"
- A major tool at Google
 - 2.2 million jobs in September 2007 (<http://goo.gl/dsnDI>)
 - In 2008 about 100k MapReduce jobs **per day**
 - over 20 petabytes of data processed per day
 - each job occupies about 400 servers

Example: Reverse Web-Link Graph



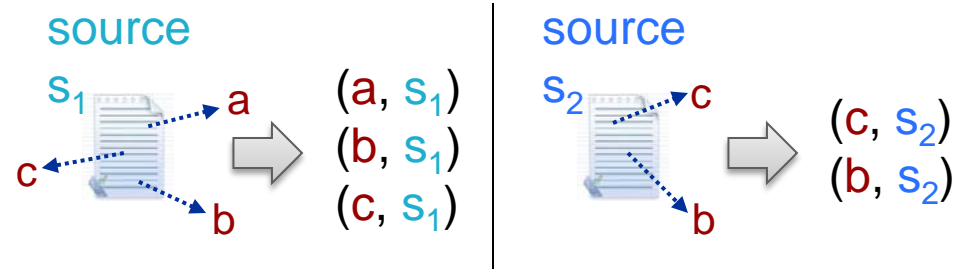
A Problem suitable for MapReduce:

In a set of html-documents, find out which other documents point (via links) to it (for each document)

Reverse Web-Link Graph: Solution

parallel MAP

1. For each link to **target** t found in document **source** emit (t, source)



2. Sort all pairs by same t 's

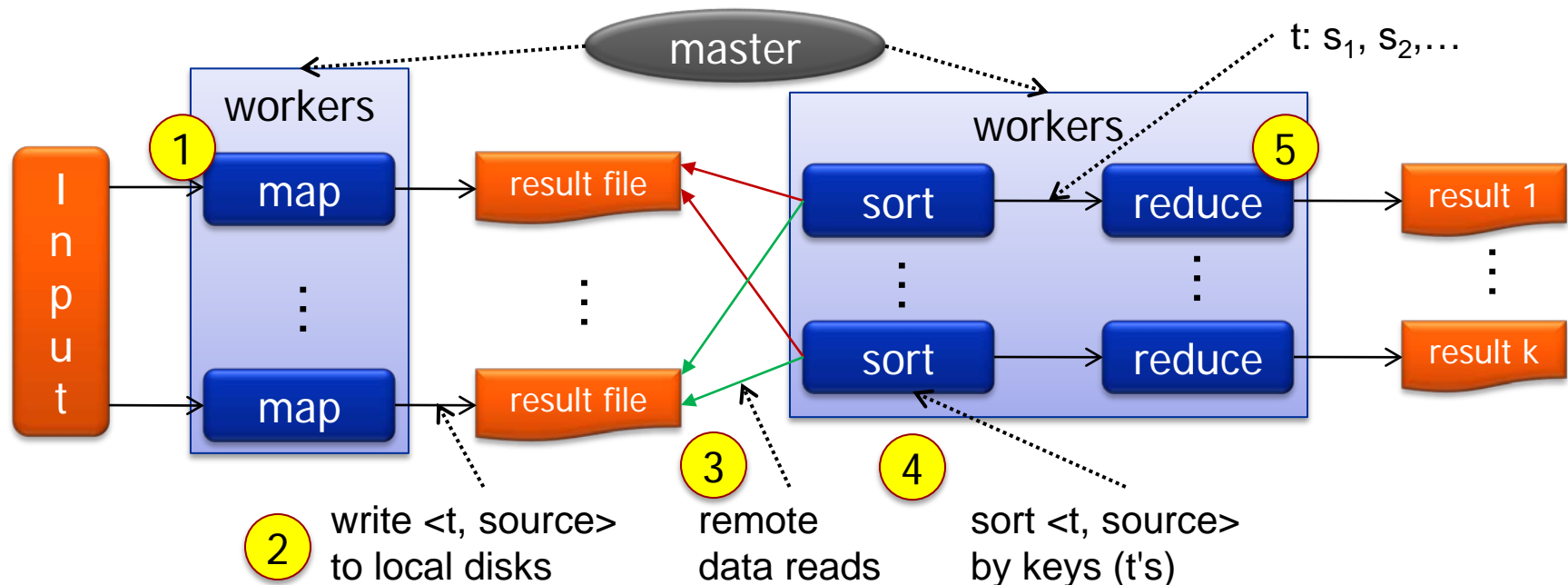
$a: (a, s_1)$
 $b: (b, s_1) (b, s_2)$ **parallel SORT**
 $c: (c, s_1) (c, s_2)$

3. Results are lists:
 $\text{list}_t(\text{source})$ for each t

$\text{list}_a(s_1)$
 $\text{list}_b(s_1, s_2)$ **parallel REDUCE**
 $\text{list}_c(s_2, s_2)$

Distributed MapReduce Frameworks

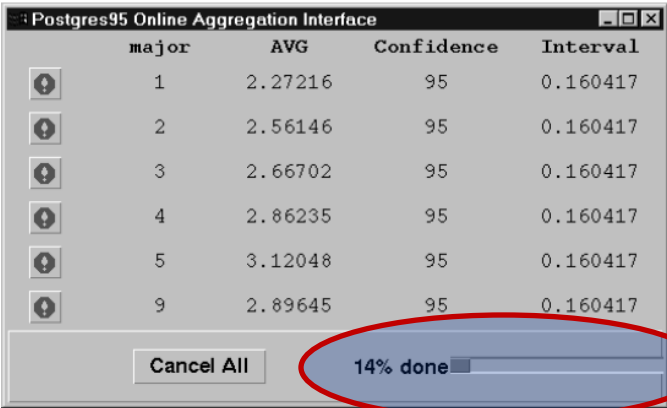
- Large data sets require **distribution**
 - e.g. 1000s of map / reduce tasks in parallel
- Not only Google: **Hadoop** is an open-source implementation



Online Aggregation

- J. M. Hellerstein, P. Haas and H. Wang introduced in 1997 the concept of **Online Aggregation**
 - Report online preliminary results (and confidence intervals) for very large queries

```
SELECT AVG(final_grade) from grades
WHERE course_name = `CS186`
GROUP BY major;
```



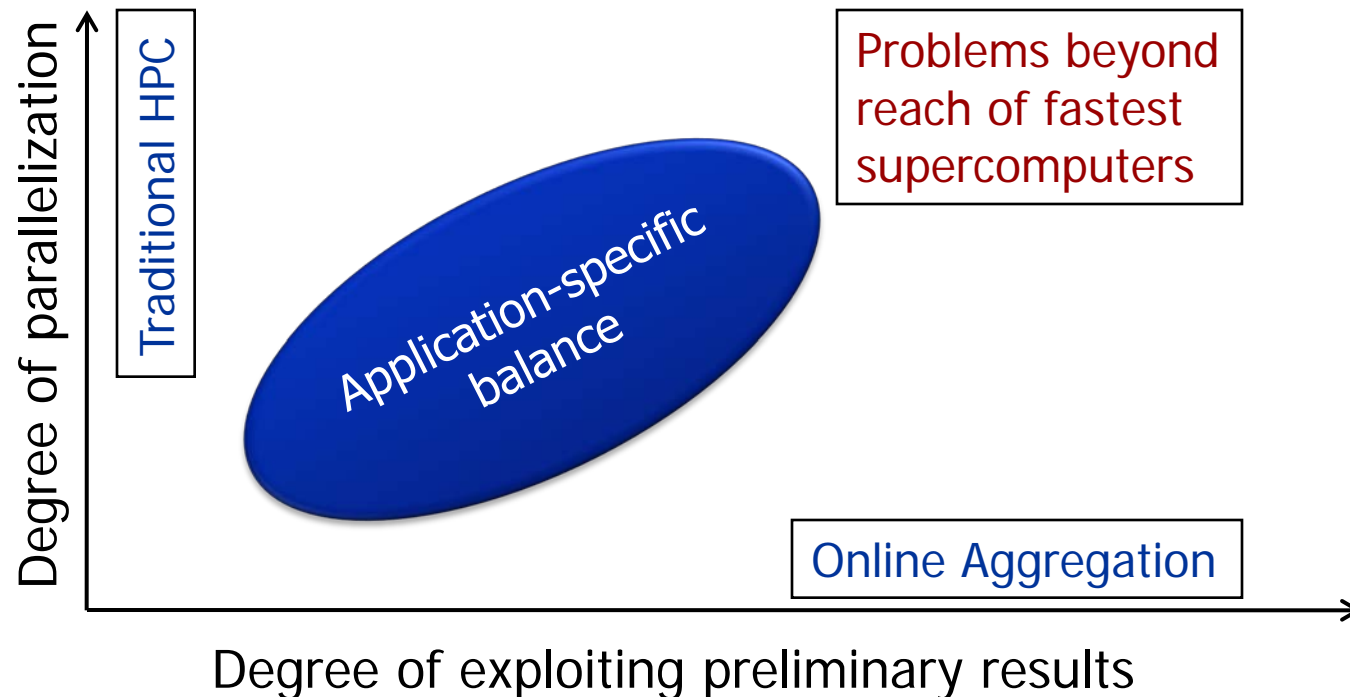
	major	AVG	Confidence	Interval
ⓘ	1	2.27216	95	0.160417
ⓘ	2	2.56146	95	0.160417
ⓘ	3	2.66702	95	0.160417
ⓘ	4	2.86235	95	0.160417
ⓘ	5	3.12048	95	0.160417
ⓘ	9	2.89645	95	0.160417

Cancel All 14% done

- **Shortens „Time-to-Decision“** in an exploratory data study
 - Allows to cancel a futile query prematurely
 - ... Or stop fast if results are precise enough
 - Helps to identify early how to drill down data

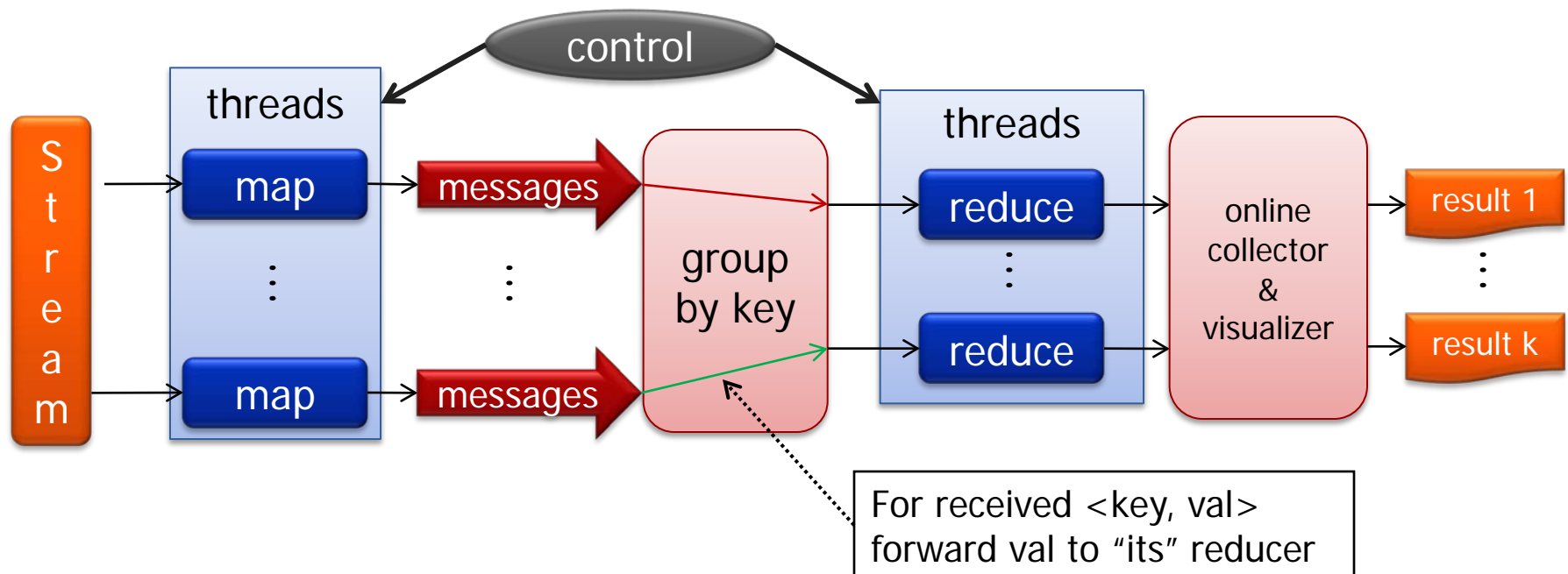
Online *Computing*

- Generalizing to **Online Computing**
 - Arbitrary computations / data analysis
 - Parallelization
- Parallelization gives two dimensions of scalability



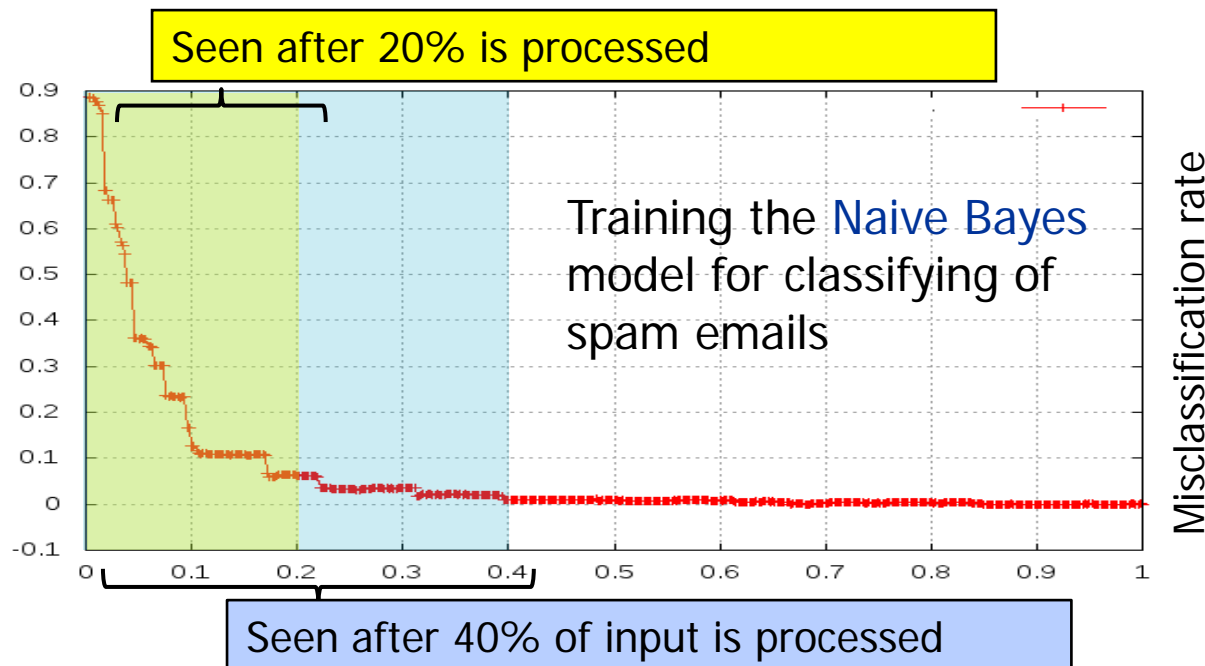
Online MapReduce

- We implemented an **streaming version of MapReduce**
 - Shared-memory; Scala + Java
 - Focus on data mining algorithms
 - Independently: a TR „Online Hadoop“ from UC Berkeley



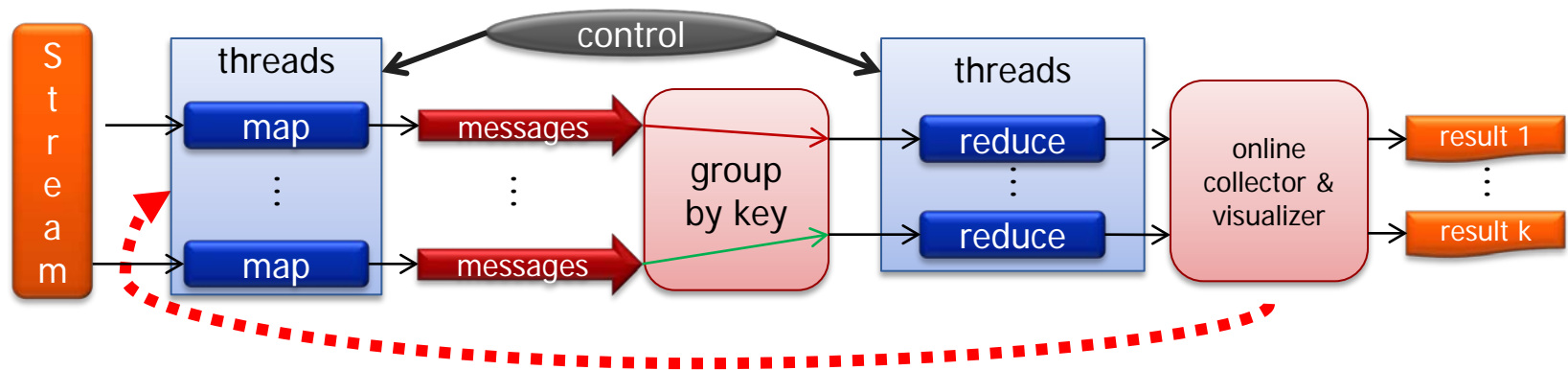
Applications to Data Analysis

- Goal: **faster deciding in data analysis studies**
 - Estimating whether preliminary solution is stable enough
 - Detecting changes in data profile
- Example: **online convergence graph**
 - Updates **periodically** the history of preliminary results



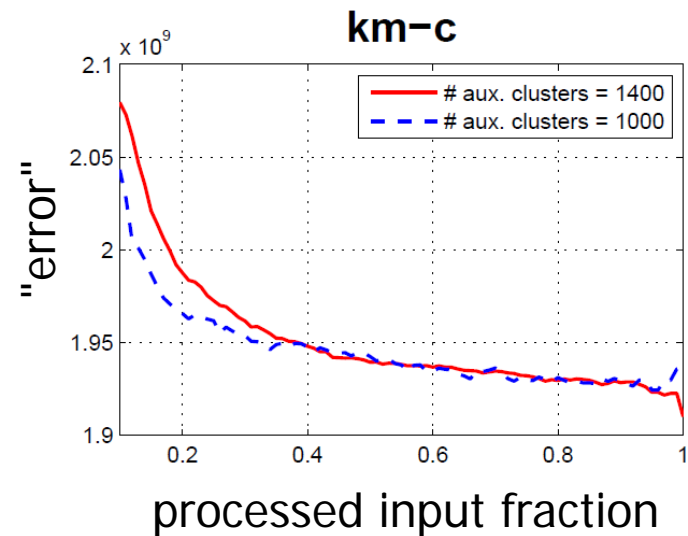
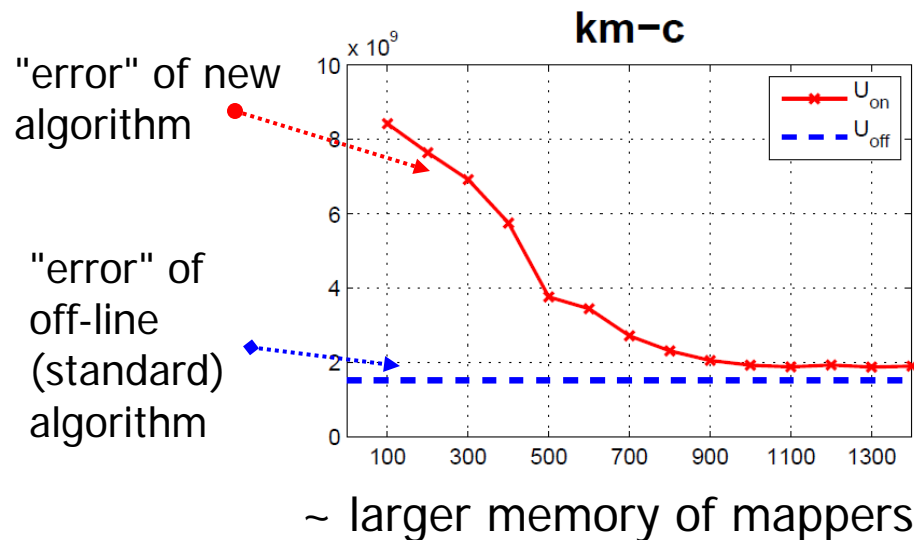
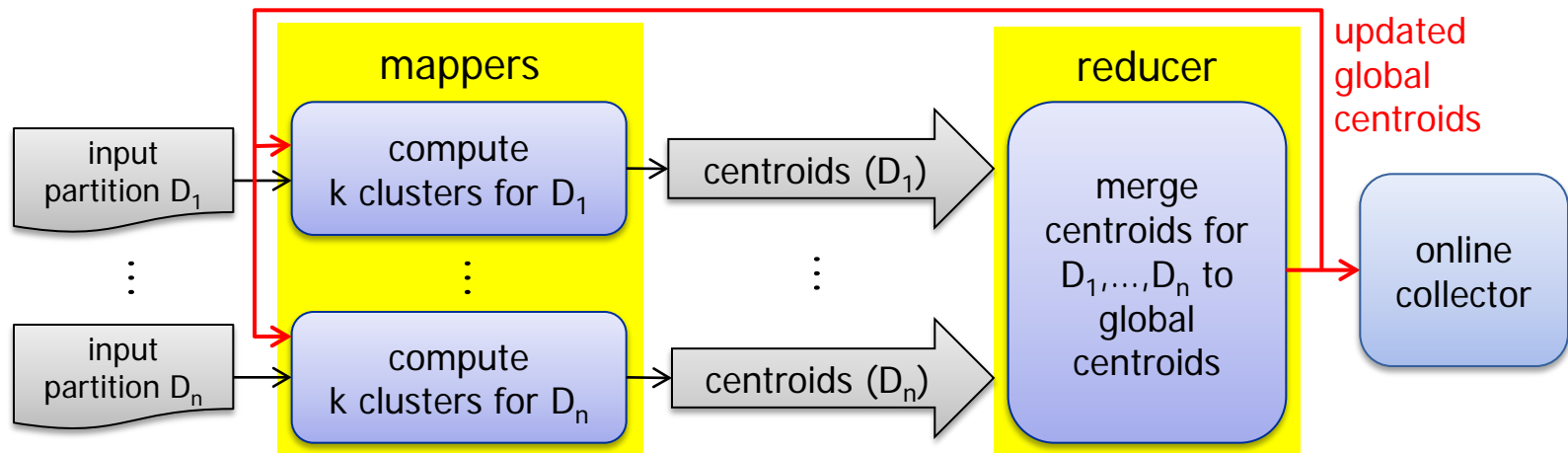
Algorithms

- Simpler algorithms require **only *one* MapReduce pass**
 - Aggregation (AVG, SUM, ...), Linear regression, PCA, Classification (Naïve Bayes), ...
- **Challenging are *multi-pass* algorithms**
 - For iterative approaches, e.g. clustering via *k-means*
- Efficiency dictates changes in algorithms / framework



Feeding back preliminary results to avoid multiple passes

K-Means Clustering Algorithm



Applications and Future

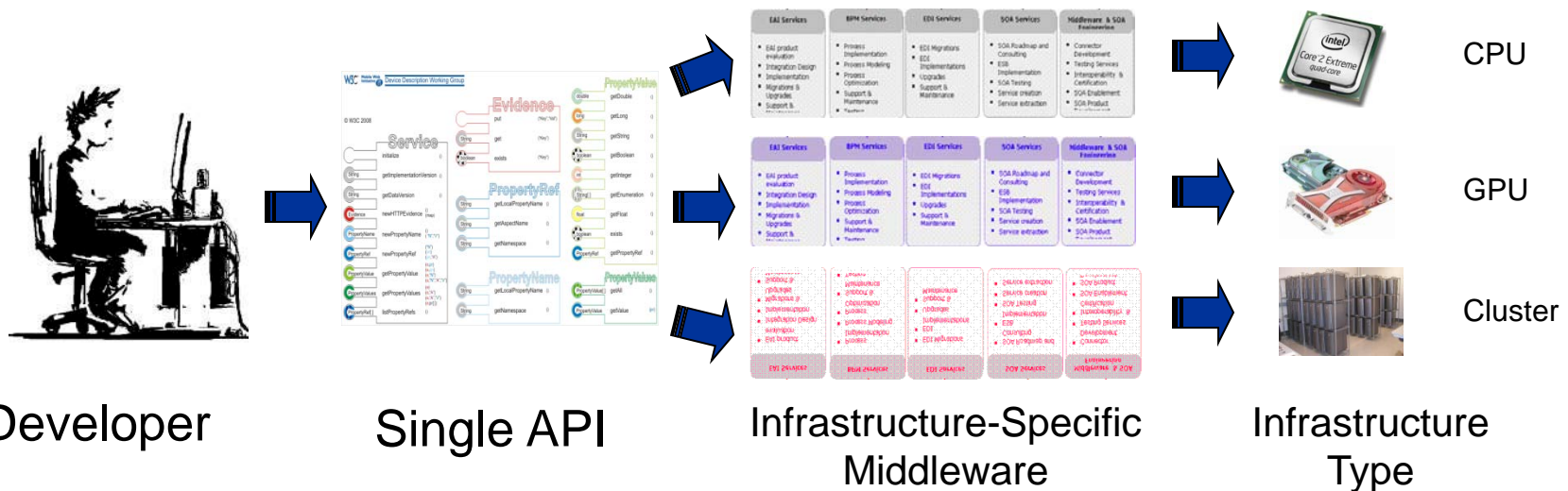
- Generic method => useful in several scientific domains
 - Analysis of very **large data sets**: astrophysics; geo-sciences, ...
 - Analysis of **data streams**: network traffic analysis; particle physics (LHC), ...
- The MapReduce model will gain **more importance for scientific computing** in the future
 - Hides parallelism – programming distributed jobs much easier
 - Prefers in-place computation with few communication phases: higher parallel efficiency - due to the “memory-wall” effect
- Incremental version with “feeding back” preliminary results facilitates porting of existing algorithms

More Research Problems

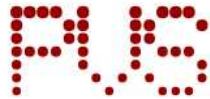
- How to **enable machine learning algorithms** to work **incrementally** (online) **and** **in parallel**?
- How to help programmers to access / **integrate MapReduce processing in only few lines of code**?
- How to **reduce the inefficiencies** of the frameworks **for smaller data sets**?
 - A job with 5 MB (linear regression) needs on Hadoop 30x longer than on a simple “ad-hoc” MapReduce simulator with 2 threads

Efficient Map-Reduce

- One API but infrastructure-dependent middleware
 - Dynamic selection of the right infrastructure depending on the input size
 - Challenges: coherent APIs and „semantics“; high implementation effort



MRStreamer



Parallel and Distributed Systems Group

Prof. Dr. Artur Andrzejak
Institute of Computer Science
Heidelberg University

UNIVERSITÄT
HEIDELBERG



Home » Software » **MRStreamer**

Home

Team

Teaching

Publications

Software

Octave scripts in KNIME

MRStreamer »

Version 0.9

Job openings

Contact

Overview

MRStreamer is a MapReduce framework implementation which provides a basic Apache Hadoop-compatible API and advanced streaming MapReduce features.

Download

Download [the latest version of the MRStreamer](#). This is the first public release and does not implement the whole Hadoop API. See the TODO list at the end of this page.

Installation and basic usage

Unpack the downloaded file on all the machines you want your MapReduce programs to run on.

Write your MapReduce program targetting either the old `org.apache.hadoop.mapred` or the new `org.apache.hadoop.mapreduce` API. Instead of using the Hadoop `JobClient` you will need to use the `MRSJobClient` and instead of `Job` `MRSJob`, respectively. Examples can be found in the `examples` subdirectory.

The machines need to have a shared filesystem such as NFS. Since MRStreamer does not implement HDFS. Run `./server` on one node and `./worker` on

Features

- Efficient shared-memory processing and “flip-a-switch” cluster processing
- Batch-mode and incremental (online) processing
- Hadoop-compatible APIs

<http://pvs.ifi.uni-heidelberg.de/software/mrstreamer/>

RESOURCES WITH HETEROGENEOUS AVAILABILITY

Hierarchy of Cheap Resources

- Commodity machines in data centers
- Amazon EC2 Spot Instances
- Institutional or privately-owned non-dedicated resources



Cheap Resources - EC2 Spot Instances

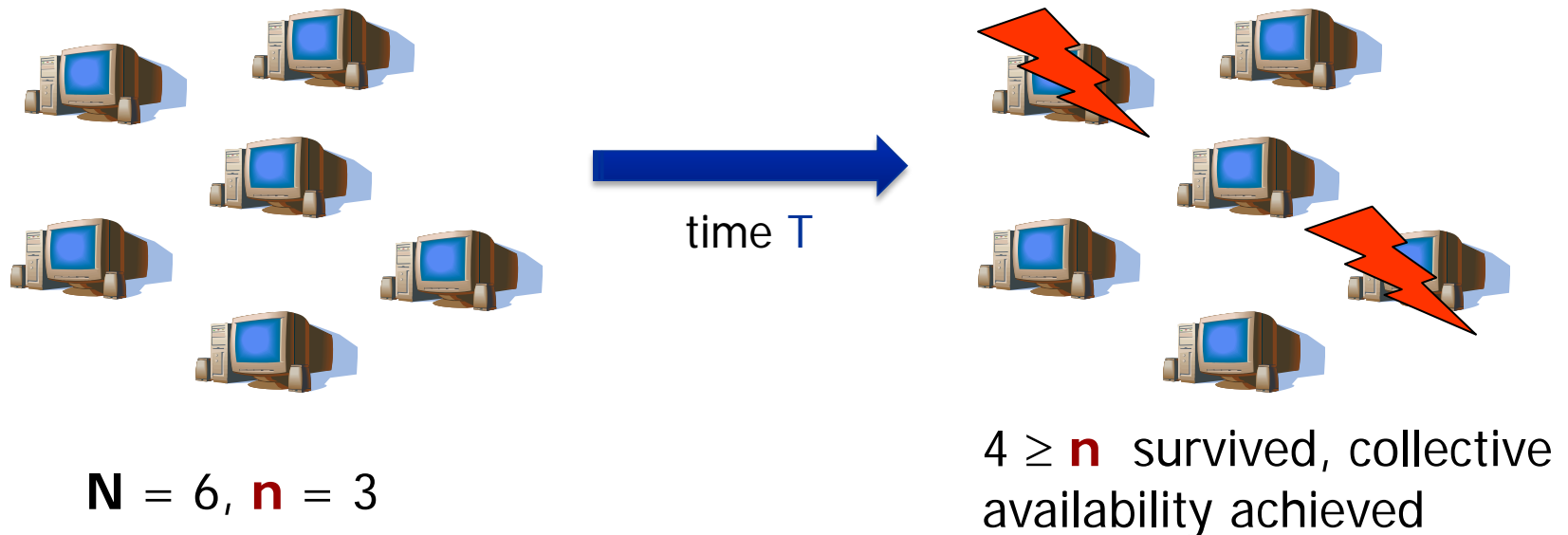
- Unused **Amazon Elastic Compute Cloud (EC2)** instances
- Price depends on user bids and fluctuates
 - Example: US - N. Virginia; Linux; on Dec 30th, 2009, 17:00 CET

Size	On-Demand Inst.	Spot Instances	Price Ratio
Small	\$0.085 per hour	\$0.026 per hour	3.3 : 1
X-large	\$0.68 per hour	\$0.265 per hour	2.6 : 1

- Users receive requested instances only their maximum bid is above the current Spot Price
- Consequence: **lower costs but jobs can be terminated without warning**

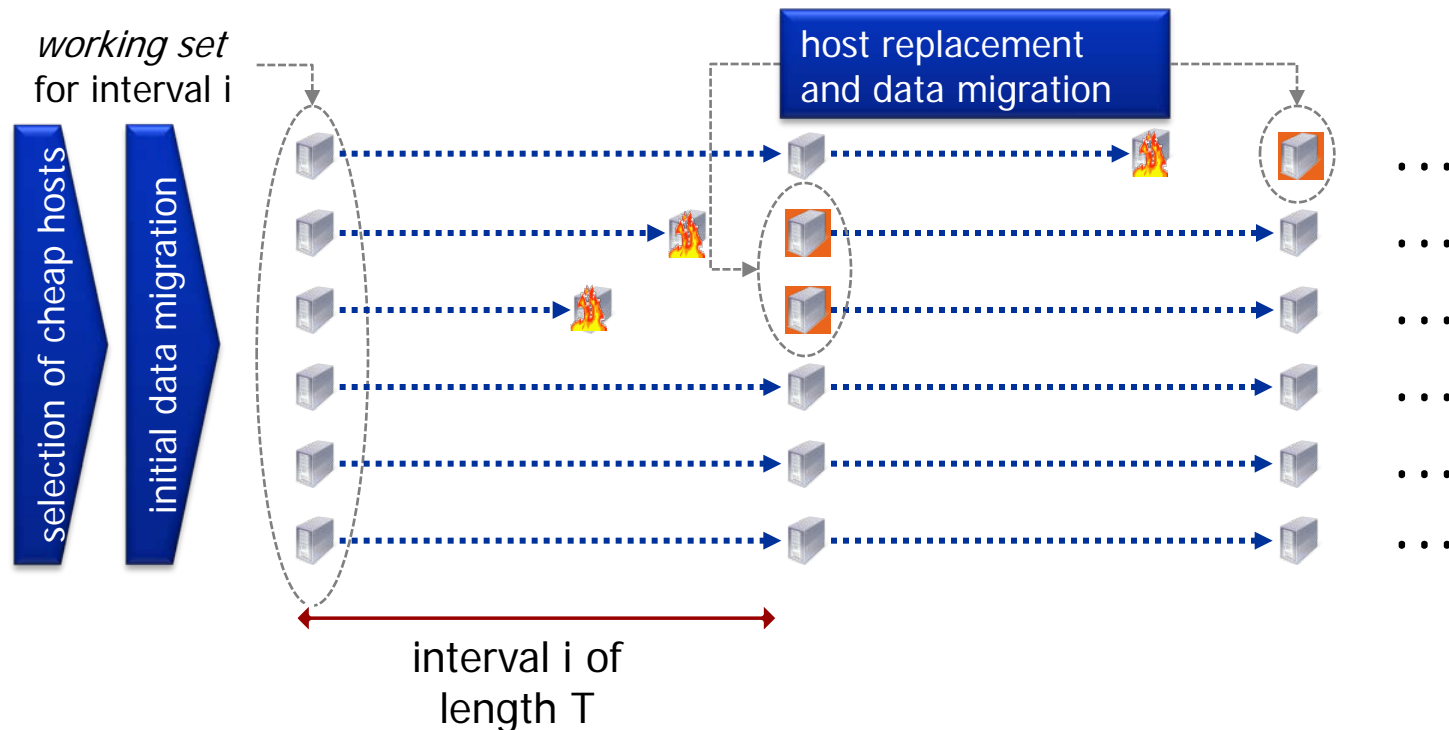
Collective Availability

- It is impossible to guarantee availability of any individual cheap resource in a given time interval
- A new concept needed: **collective availability**
 - For given **n** and **T**, a group of $N \geq n$ resources achieves *collective availability* if at least **n** of these resources are still available after time interval **T**



Ensuring Long-Term Operation

- We **replace failed hosts periodically**
- This causes costs due to
 - data migration, re-computations since last checkpoint, execution delay, ...
- Measured by **migration rate** := average (# failed resources / N)

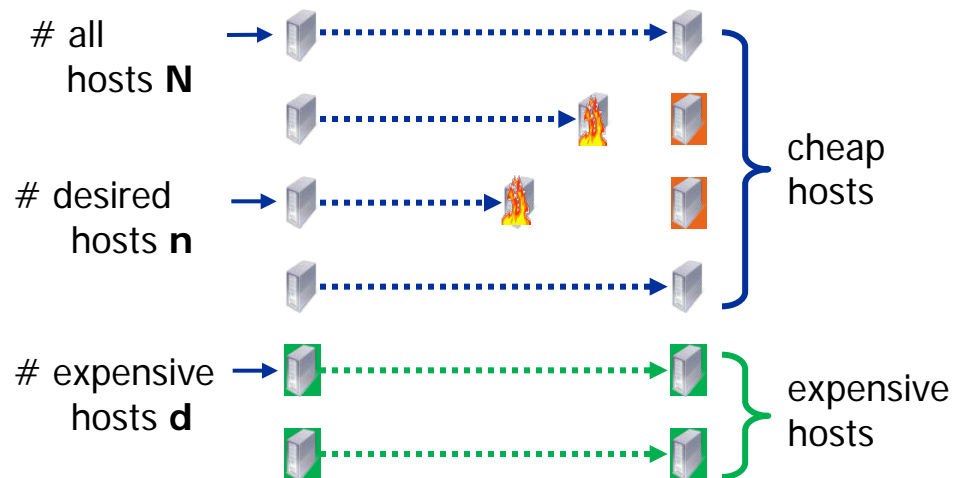


Guarantees on Collective Availability

- Benefit: we can give **statistical guarantees on collective availability**
 - We control N (the total size of the resource group) and thus the redundancy of cheap resources
- **Problem A**: Given n , T , and p – the availability level, what is the minimum total group size N to ensure collective availability with probability $\geq p$?
 - Probability is understood as the average number of successes over many time intervals

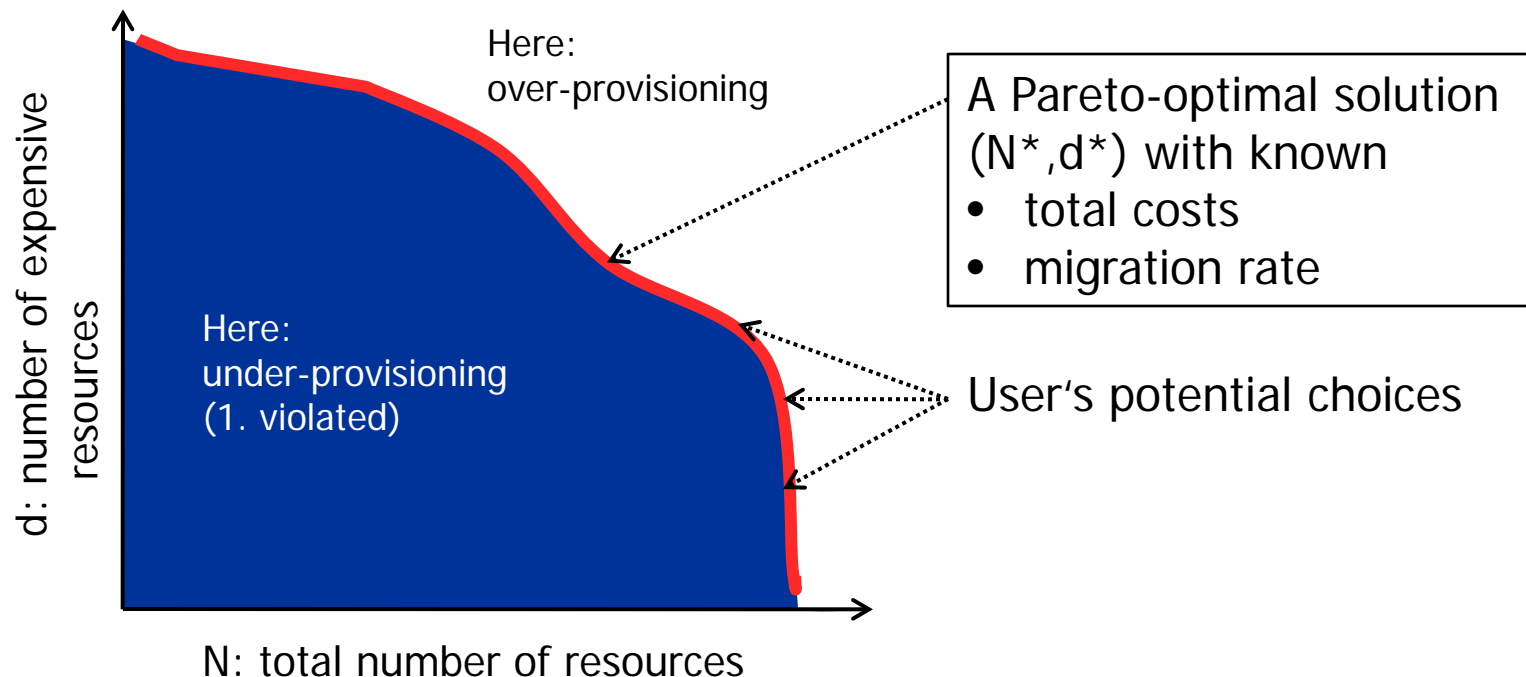
Combining Resource Types

- We include a set of **d highly available** (and possibly expensive) resources
 - Cheap resources are a complement for expensive ones anyway
- This allows to **balance cost against migration rate**
- **Trade-offs** (n and p fix)
 - Larger N: **lower costs** but **higher migration rate**
 - Larger d: **higher costs** but **lower migration rate**



Pareto-Optimal Solutions

- We provide a user with a set **Z** of **Pareto-optimal solutions (N,d)** such that
 1. Each (N,d) ensures collective availability for given n, p, T
 2. Neither N nor d can be decreased (with other parameter constant) without violating 1.

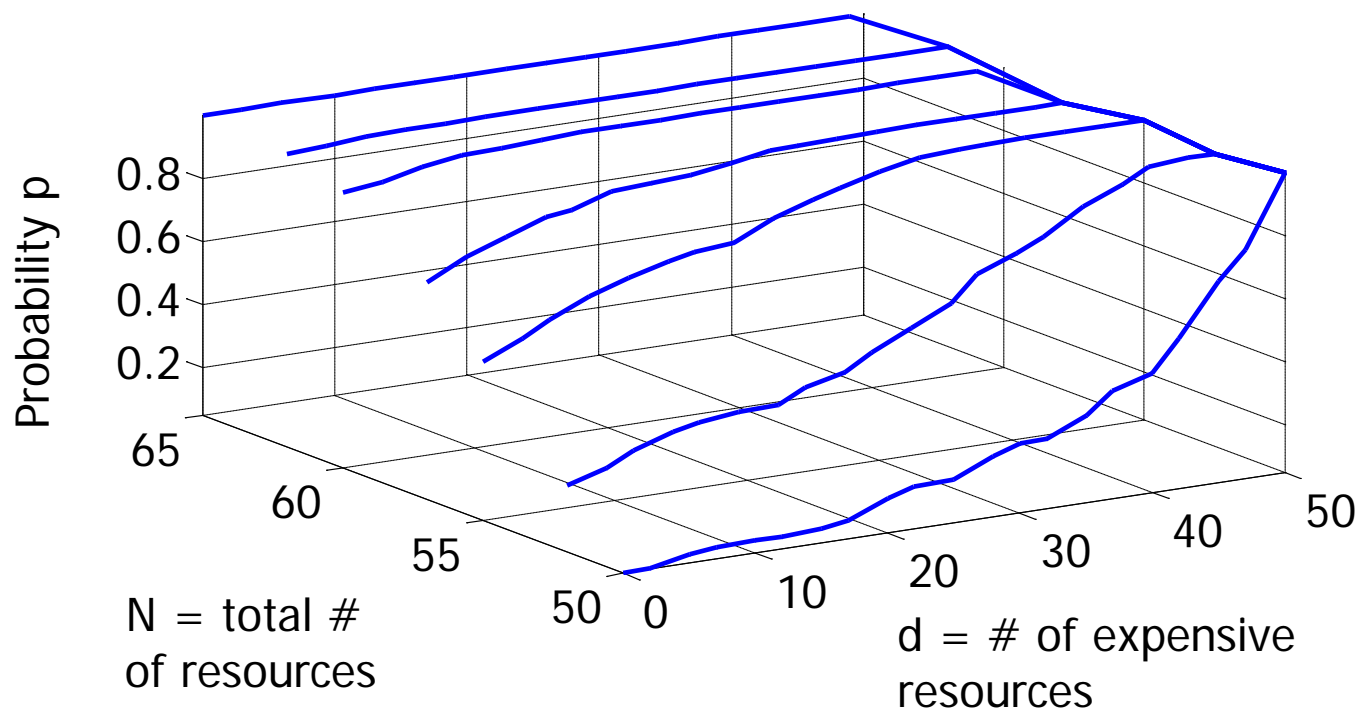


Empirical Study

- We have conducted **empirical study using real availability data** from ~10,000 SETI@home hosts
 - Collected from Dec 2007 and Feb 2008 via **BOINC clients**
- Extreme case: least available type of cheap resources
 - No data on EC2 Spot Instances, but the method would be exactly the same
- Assumptions
 - Cost of cheap resources: \$0
 - Cost of expensive resources: \$0.10 per hour (~ Amazon's EC2)

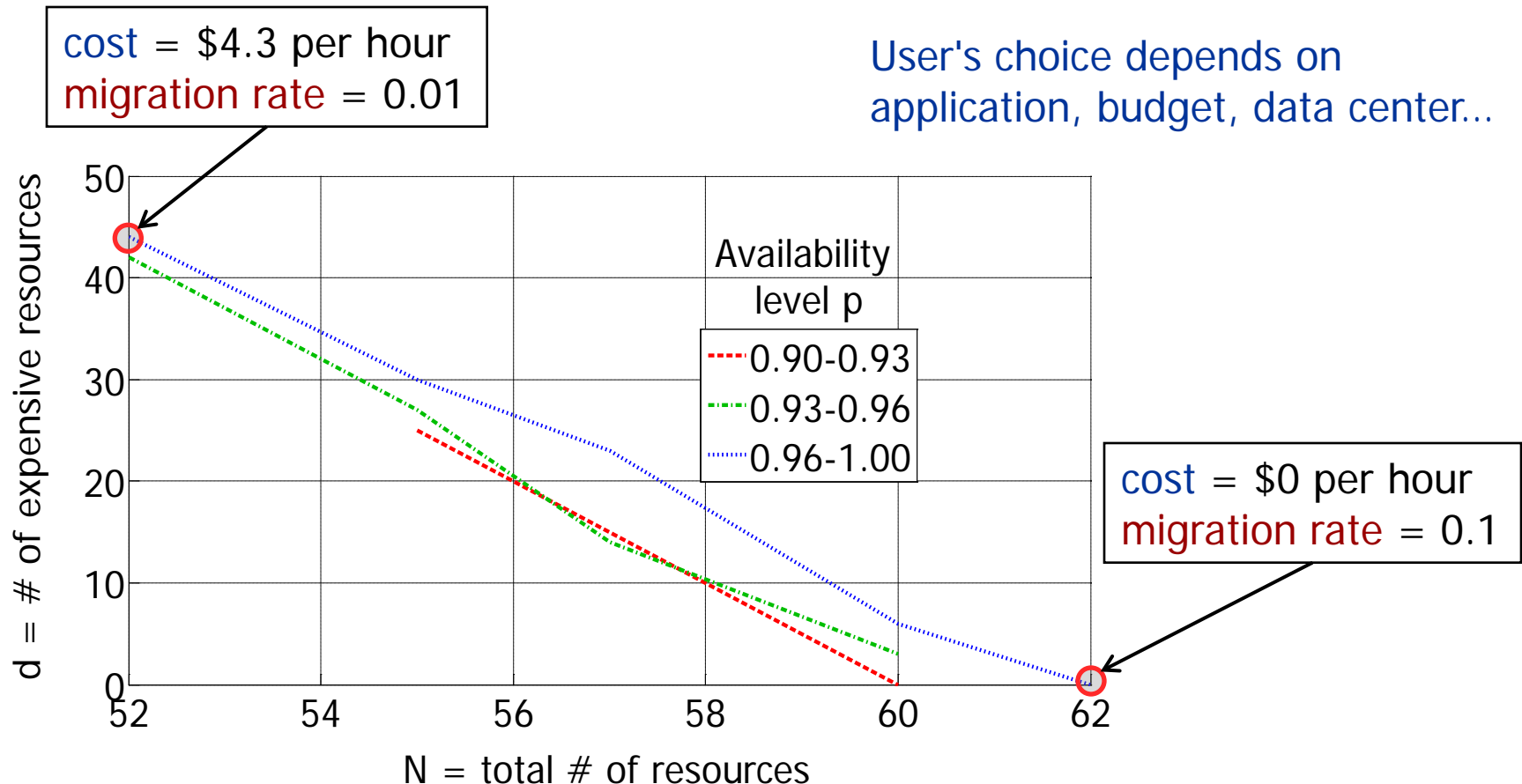
Simulation of Availability

- **Probability p of achieving collective availability**
 - For different (N, d) combinations and $n = 50$ desired resources



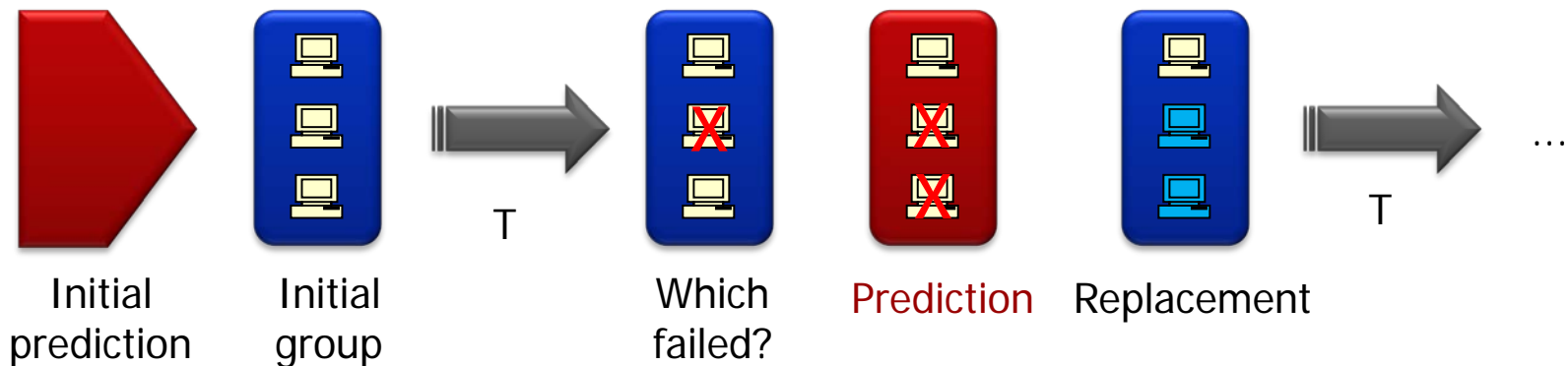
Pareto-Optimal Solutions (N,d)

- For different availability levels p
 - Recall: migration rate := average of (# failed resources / N)



Selection of Individual Resources

- **Select hosts which are most likely to be available in the next time interval**
- Short-term **prediction of availability**
 - A trivial technique turned out as the best: use last availability state as the prediction for next T hours



Conclusions

- Exploratory scientific computing and large-scale data analysis profits from **reduced Time-to-Solution**
- There are **approaches with better cost-efficiency** than increasing processor count
- Deployment depends on specific application
 - Sensitivity of the application to resource failures
 - Metrics to estimate quality of a preliminary solution
- There are several **non-trivial research challenges**
 - Parallel *and* incremental machine learning algorithms

References

- Joos-Hendrik Böse, Artur Andrzejak, Mikael Höggqvist: *Beyond Online Aggregation: Parallel and Incremental Data Mining with Online MapReduce*, [ACM MDAC 2010](#), Raleigh, NC, 2010.
- Artur Andrzejak, Derrick Kondo, David P. Anderson: *Exploiting Non-Dedicated Resources for Cloud Computing*, [12th IEEE/IFIP Network Operations and Management Symposium \(NOMS\)](#), Osaka, 2010.
- Derrick Kondo, Artur Andrzejak, David P. Anderson: *On Correlated Availability in Internet-Distributed Systems*, [9th IEEE/ACM International Conference on Grid Computing \(Grid\)](#), Tsukuba, 2008.
- Artur Andrzejak, Derrick Kondo, David P. Anderson: *Ensuring Collective Availability in Volatile Resource Pools via Forecasting*, [19th IFIP/IEEE Distributed Systems: Operations and Management \(DSOM\)](#), Samos, 2008.

THANK YOU.
QUESTIONS?

ADDITIONAL SLIDES

ONLINE COMPUTING

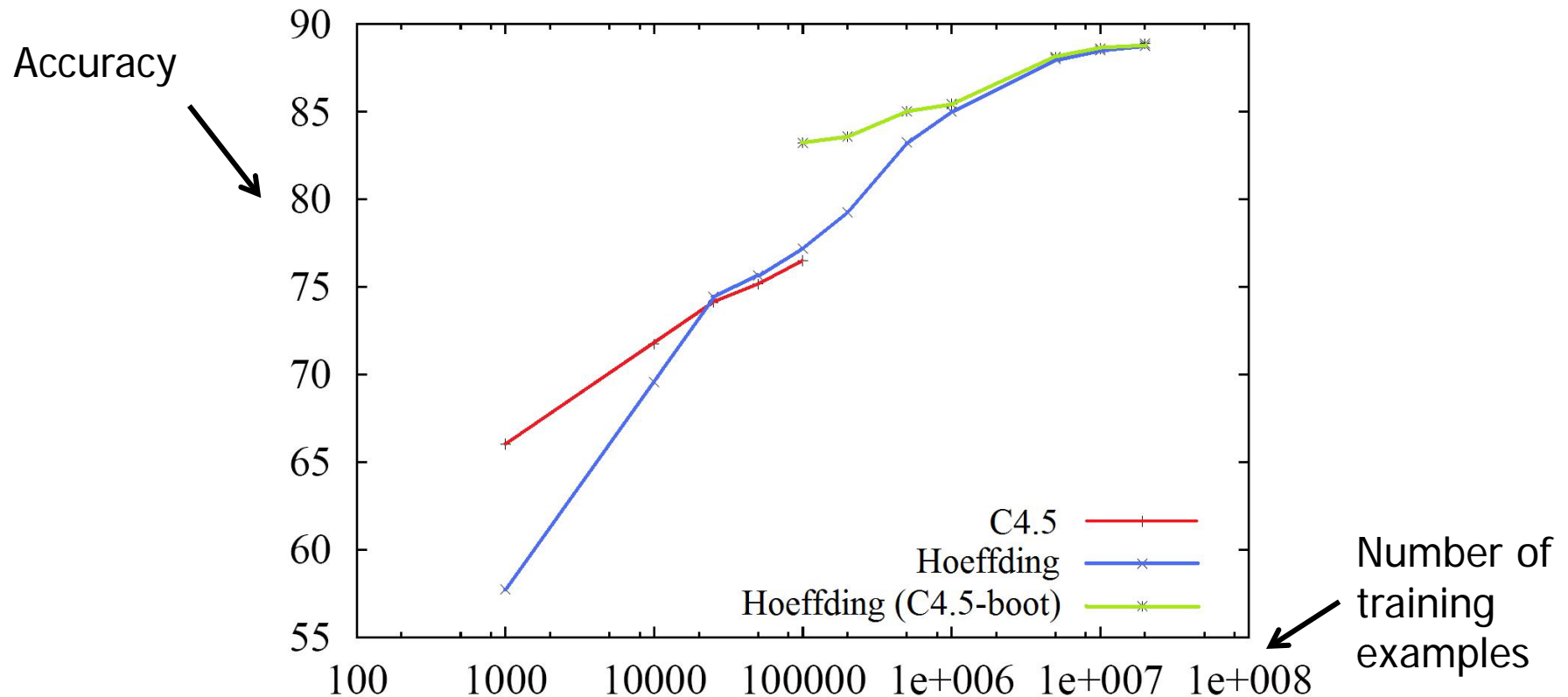
MapReduce Programming Model

- Re-discovered by Google with goals:
 - "Reliability has to come from the software"
 - "How can we make it easy to write distributed programs?"
- „**Scalable**“ in the ratio of faults and in the degree of parallelism
- A major tool at Google
 - 29k jobs in August 2004 and 2.2 million in September 2007
 - In 2008 about **100k MapReduce jobs per day**
 - each occupies about 400 servers
 - takes about 5 to 10 minutes to finish

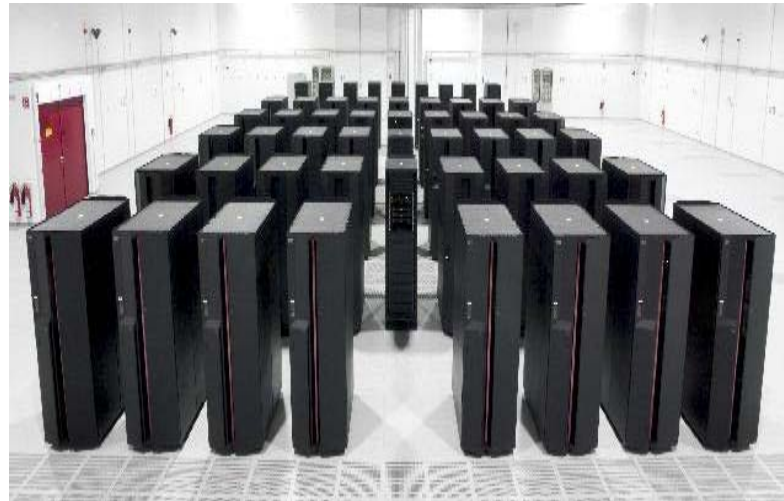
Very Fast Decision Trees

- ⊗ Domingos und Hulten (2000): **Hoeffding-Trees** (VFDT, Very Fast Decision Trees)
- ⊗ Creates trees incrementally
- ⊗ Uses **Hoeffding inequality** to decide how to extend a tree by splitting dataset at a leaf
 - ⊗ It bounds (with high prob.) the error of new decision node compared to the tree using all data ("off-line tree")
- ⊗ Properties
 - ⊗ Need only one pass through data and memory ~ to tree size
 - ⊗ Very fast – I/O transfer is the bottleneck
 - ⊗ Asymptotically H-trees become same as off-line trees

Comparizon Hoeffding (VFDT) vs. C4.5



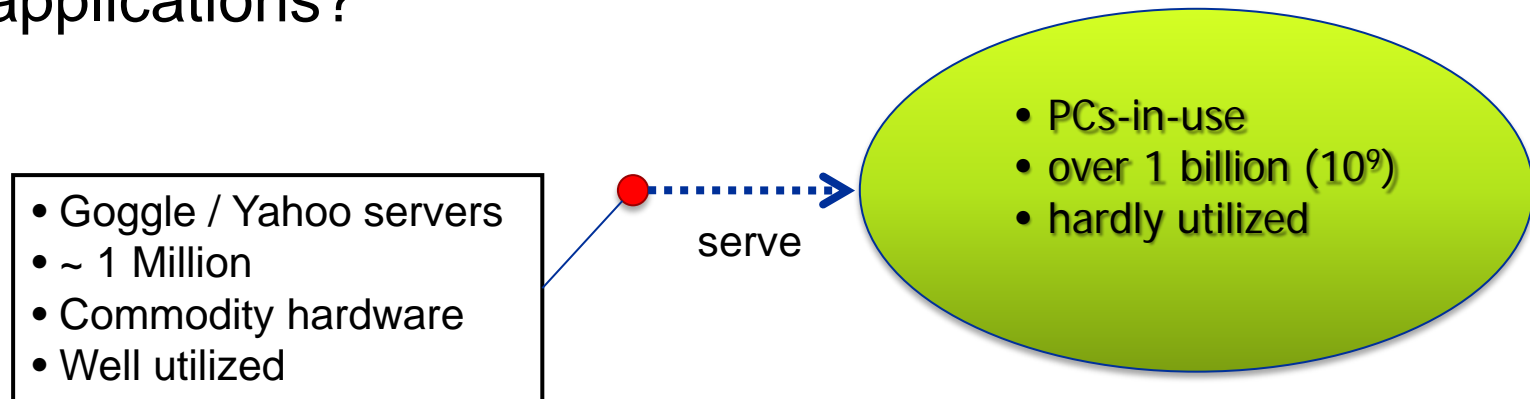
- ⊙ Artificial data (100 attributes, 2 classes) (Domingos und Hulten 2000)
- ⊙ Memory requirements of VFDT's were never larger than of C4.5



RESOURCES AND COST: VOLUNTARILY COMPUTING

Cost of Resources

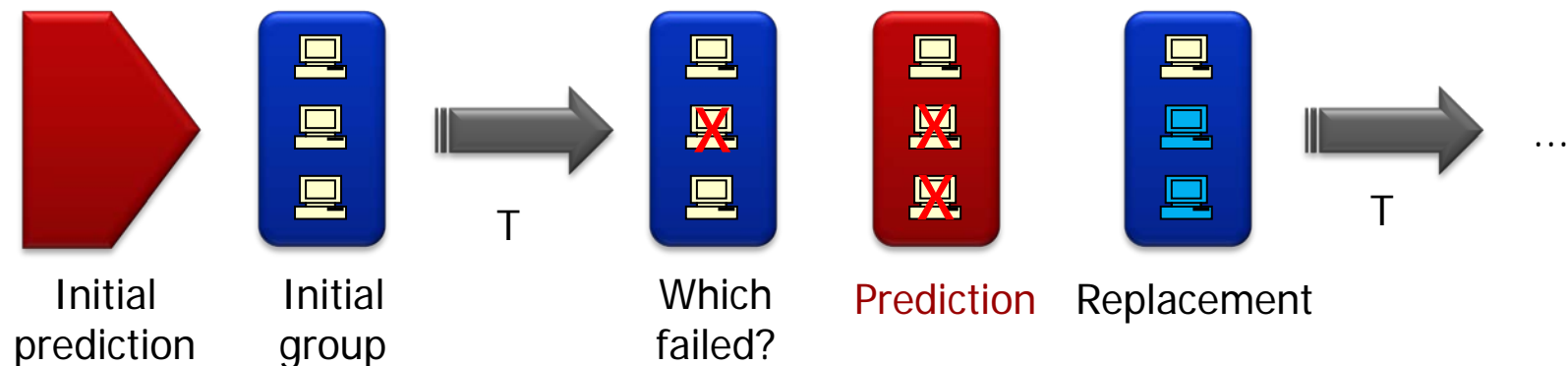
- Where is the second-fastest computer in the world?
- *Almost everywhere*: home / office PCs participating in **voluntarily computing** provide largest computing power
 - Folding@home: 6.5 petaflops as of Oct 2011
- K computer RIKEN (AICS): 8.7 PF (Jun 2011)
- Can we harness these resources for service-type (SOA) applications?



PREDICTING AVAILABILITY

Selection of Individual Resources

- We **use SETI@home hosts which are most likely to be available in the next time interval**
- Two complementary methods
 1. Ranking by the „regularity” of past availability behavior
 2. Short-term prediction of availability
 - Trivial technique turned out as the best: use last availability state as the prediction for next T hours



Filtering Hosts

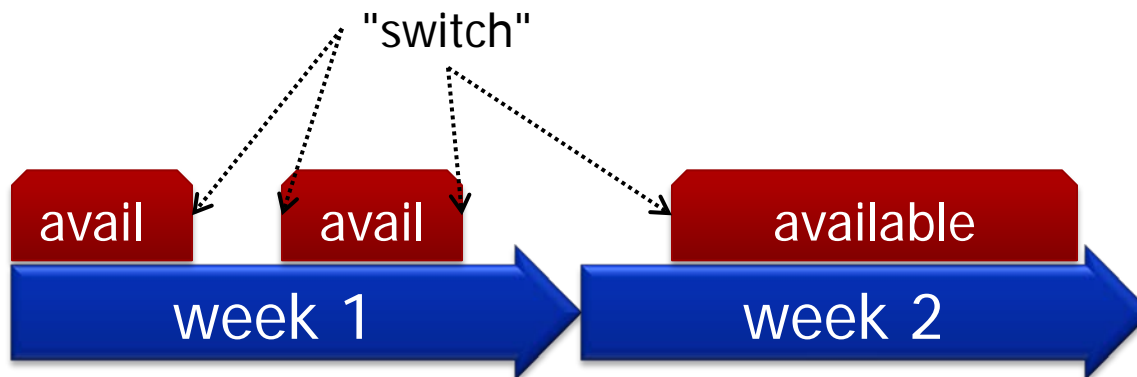
- We want to find out, for each host, whether its availability predictions are likely to be accurate
- We want hosts with high predictability:
 - def.: *expected accuracy of predictions from a model build on historical data*
- To estimate it, we use **indicators of predictability**
 - fast to compute
 - use only training data

A. Filter hosts by predictability => create a pool of "good" hosts

B. Make predictions only for hosts from the pool

Predictability Indicators

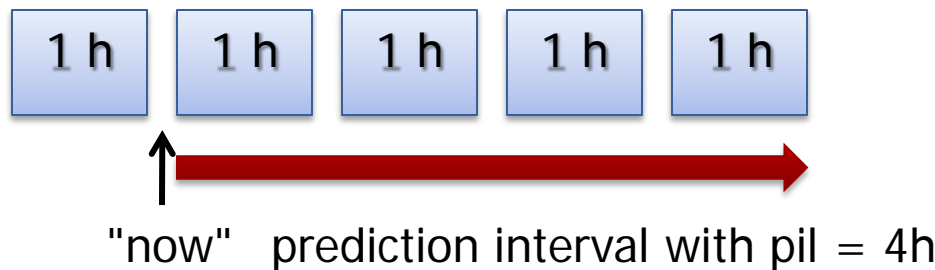
- We have tested, among others:
 - Average length of an uninterrupted availability segment
 - Size of the compressed availability trace
 - Prediction error tested on a part of the training data (as a control indicator)
 - Number of availability state changes per week (**aveSwitches**)



$$\text{aveSwitches} = 4 / 2 = 2$$

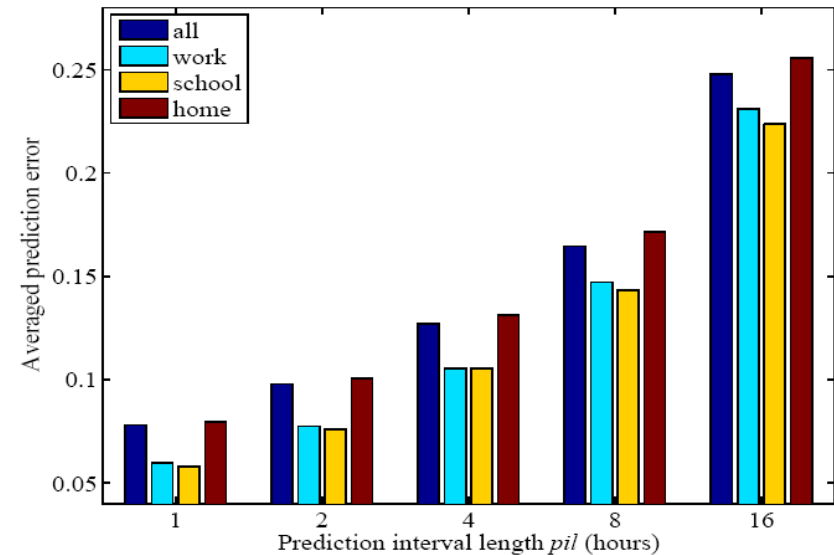
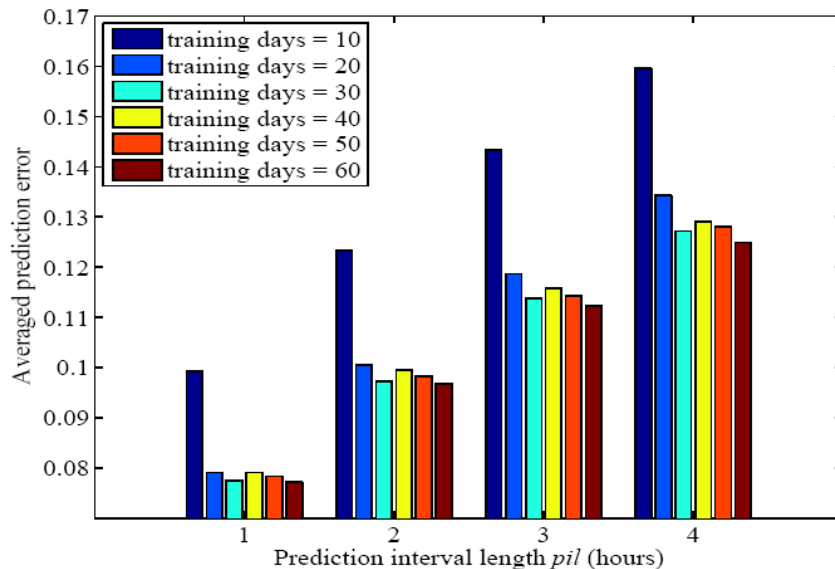
Prediction

- A. Classifiers using models built on historical data
 - Vectors of "raw" availability + preprocessed data over 30 days
 - B. Modeling on/off-intervals by Normal distribution
 - C. "TakeLast": take last known availability, and project it
- Predictions for each hour over two weeks
 - starting now, will the host be available in the next k hours
 - this is **prediction interval length, pil**



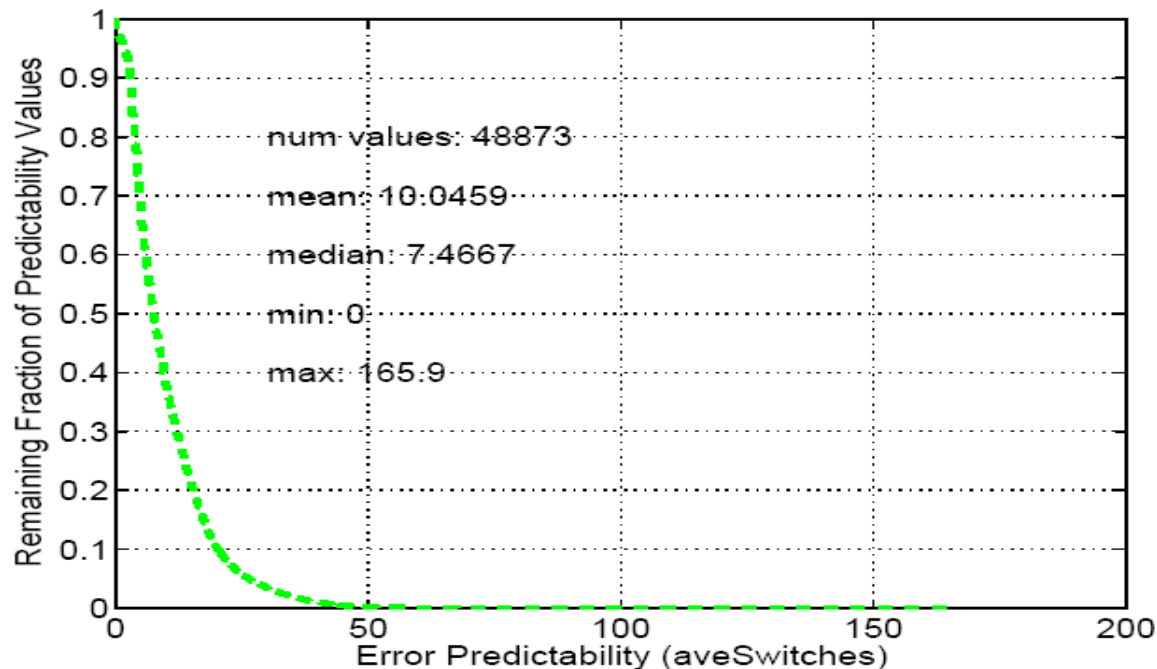
What drives accuracy?

- Dependence upon
 - prediction interval length, pil
 - training interval length
 - host ownership type (private, school, work)



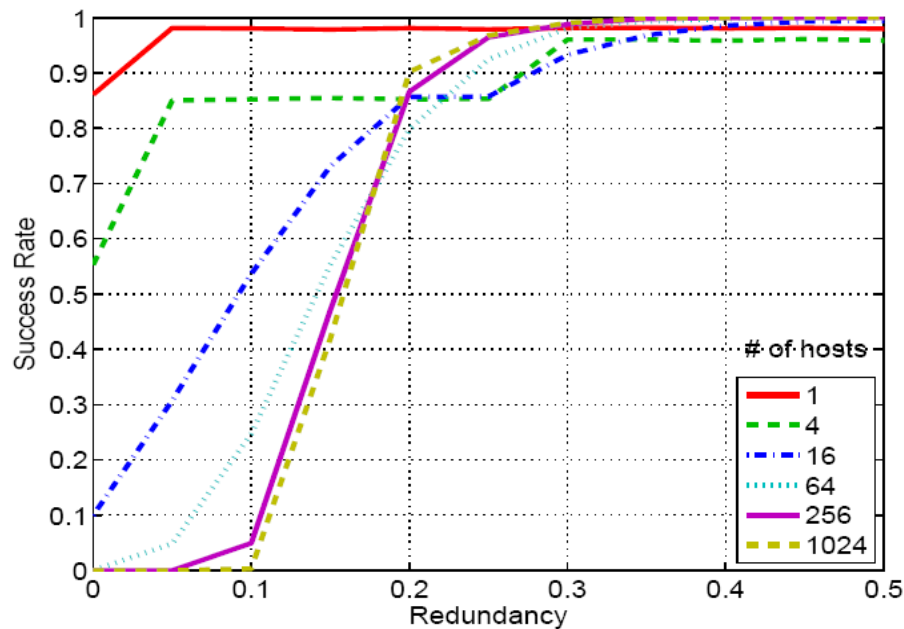
The Role of Predictability

- A little twist - we select R hosts from two groups:
 - low predictability, with aveSwitches ≥ 7.47
 - high predictability, with aveSwitches < 7.47

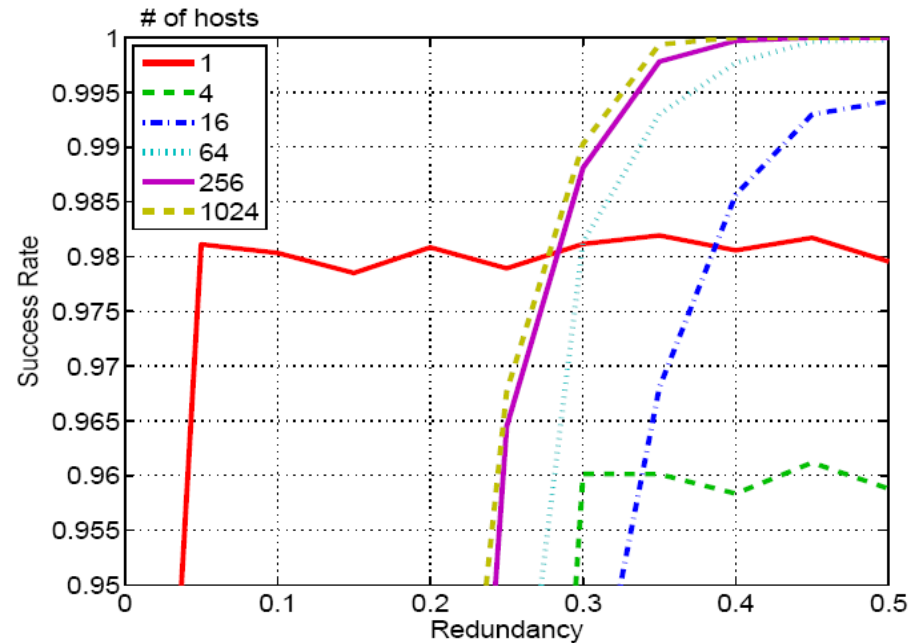


Necessary Redundancy

- **High** predictability group ($\text{pil}=4$)



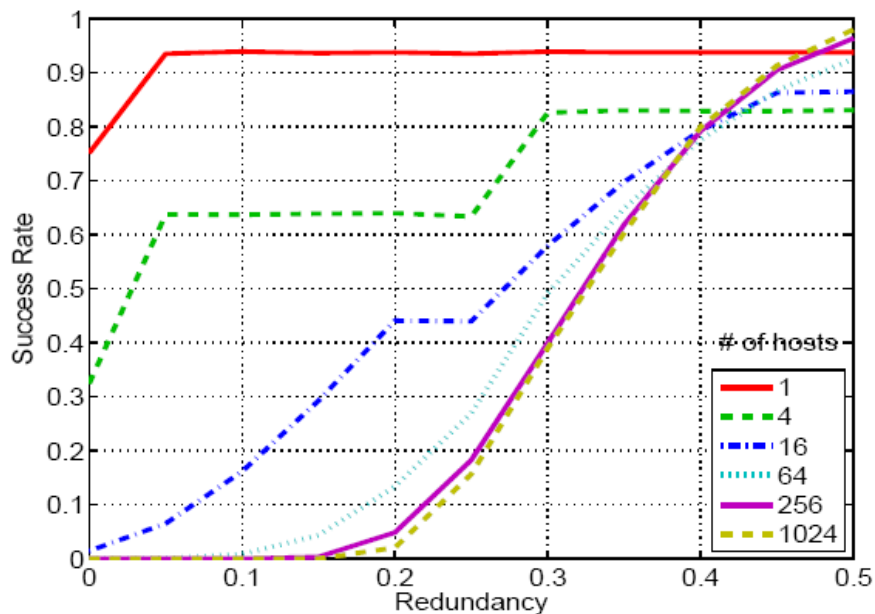
(a) Complete range



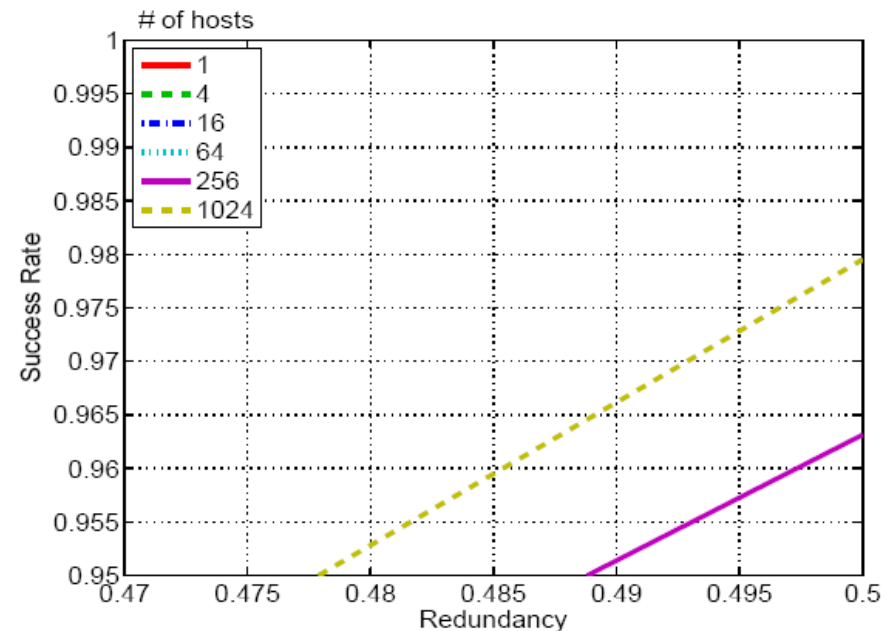
(b) Zoomed-in range

Necessary Redundancy

- **Low** predictability group ($\text{pil}=4$)



(a) Complete range



(b) Zoomed-in range

Is this Redundancy too high?

- In high predictability group, we have required **redundancy of 35%**
 - One of the reviewers called this factor "disappointingly high"
- However, we consider this dramatically low
 - In comparison, SETI@home has 200% redundancy (also used for result validation)
 - In terms of absolute savings, that equates to 165 TeraFLOPS saved in a system such as FOLDING@home => **significant power savings**

COLLECTIVE AVAILABILITY

Hierarchy of Cheap Resources

- **Commodity machines** in data centers
 - Problems due to large number of unreliable boxes
- **EC2 Spot Instances**
 - Indirect availability control (by bid price)
- **Institutional / privately-owned non-dedicated resources**
 - Cycles donated by volunteers / office PCs
 - No control of availability at all



Computing Pareto-Optimal Solutions

- How to **compute all Pareto-optimal (N,d) 's**?
- We assume that none of d expensive resources ever fail
 - Then n = desired number of resources becomes $n' := n - d$
- If d fixed, **computing (N,d) 's reduces to Problem A**:
 - Given n' , T , and p , what is the minimum total size N to achieve collective availability with probability $\geq p$?
- We compute p for combinations of various n , N , d values
 - **Simulate** many requests of n desired hosts over T hours intervals
 - Probability $p = (\# \text{ of trials with col. avail. achieved}) / (\text{all trials})$
 - Group solutions by similar p