Stratosphere – Data Management on the Cloud

Odej Kao
Complex and Distributed IT Systems
Computer Science and Electrical Engineering
Technische Universität Berlin

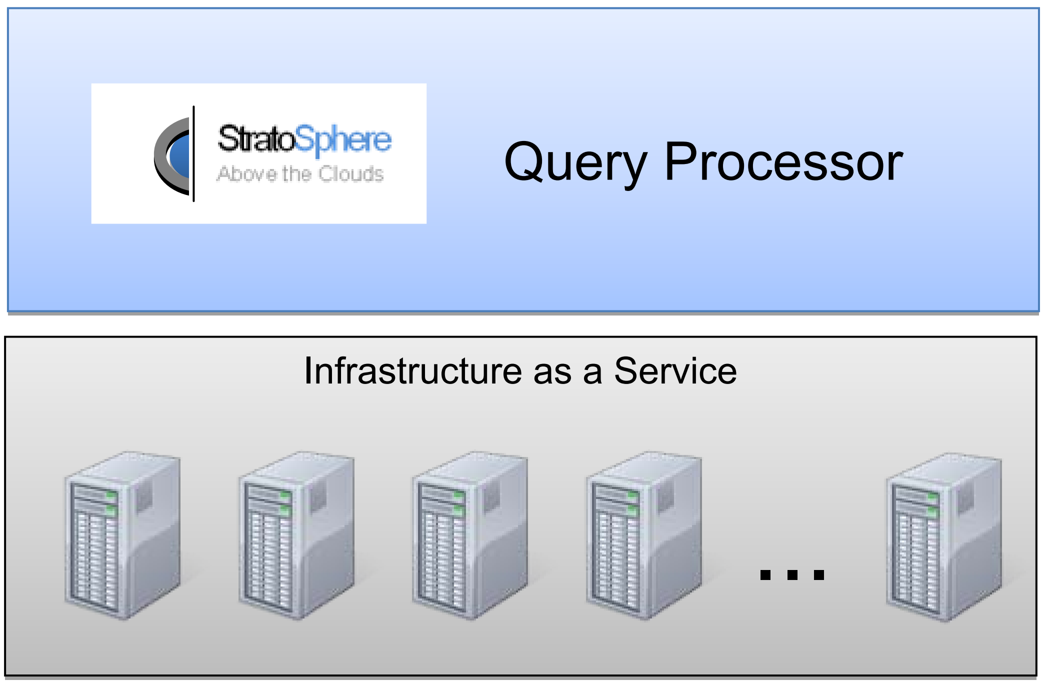
This presentation is a joint work with Volker Markl, Andreas Kliem, Björn Lohrmann and Daniel Warneke

StratoSphere
Above the Clouds

Fraunhofer FIRST

Stratosphere

Explore the power of Cloud computing for complex information fusion



- Database-inspired approach
- Analyze, aggregate, and query
- Textual and (semi-) structured data
- Research and prototype a web-scale data analytics infrastructure

Current Research Landscape

- Large scale data management is area of vivid research
  - Google, Yahoo!, Microsoft, Facebook, IBM, UC Berkeley, UC Irvine, etc.

Higher-Level Language	Pig, Jaql, Hive	Scope, DryadLINQ	AQL, Pig, Hive	SIMPLE/ Sopremo
Parallel Programming Model	Map/Reduce		Algebricks	PACT
Execution Engine	Hadoop	Dryad	Hyracks	Nephele
	Hadoop Stack (Yahoo!, Facebook)	Dryad Stack (Microsoft)	Asterix Stack (UCI, UCR, UCSD)	Stratosphere Stack (TU, HU, HPI)

Outline

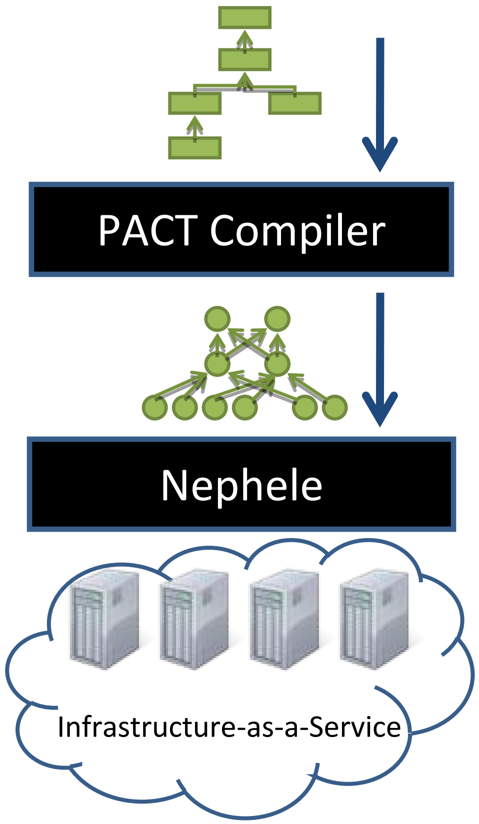
- Overview Stratosphere
- Massive-parallel execution with Nephele
- Topology detection and streaming
- Conclusions

Stratosphere in a Nutshell

- PACT Programming Model
  - Declarative definition of data parallelism
  - Centered around second-order functions

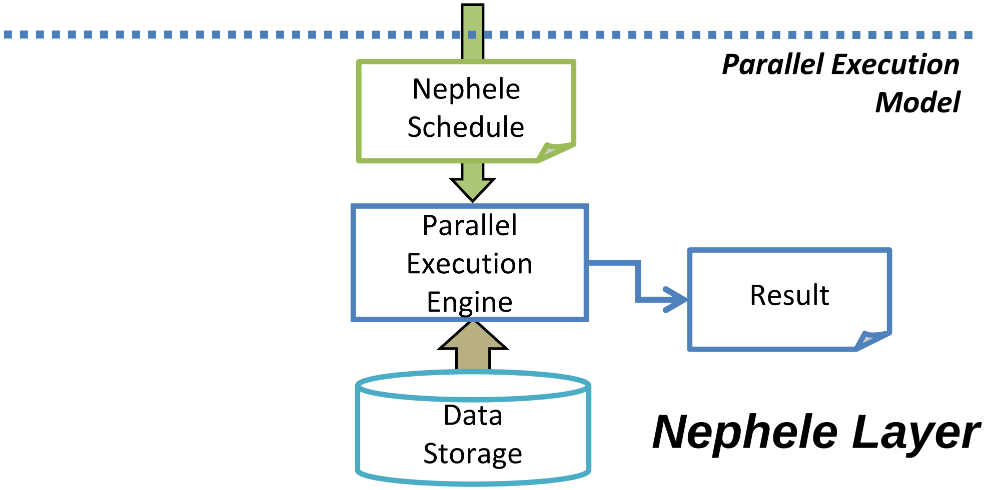
⇒ Generalization of map/reduce

- Nephele
  - Executes schedules compiled from PACTs
  - Exploits scalability/flexibility of clouds
  - Fault tolerance mechanisms
  - Designed to run on top of IaaS
  - Heterogeneity through different VM types



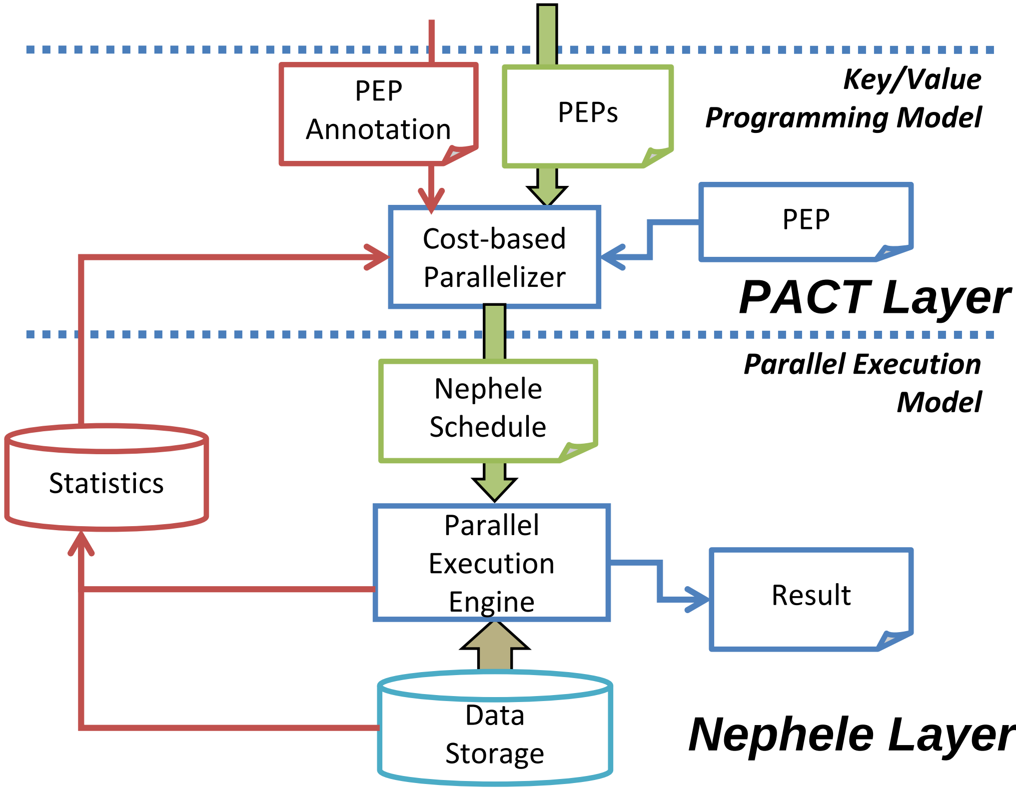
Architecture: Nephele Layer

- Key Concepts
  - Massively parallel, fault-tolerant engine



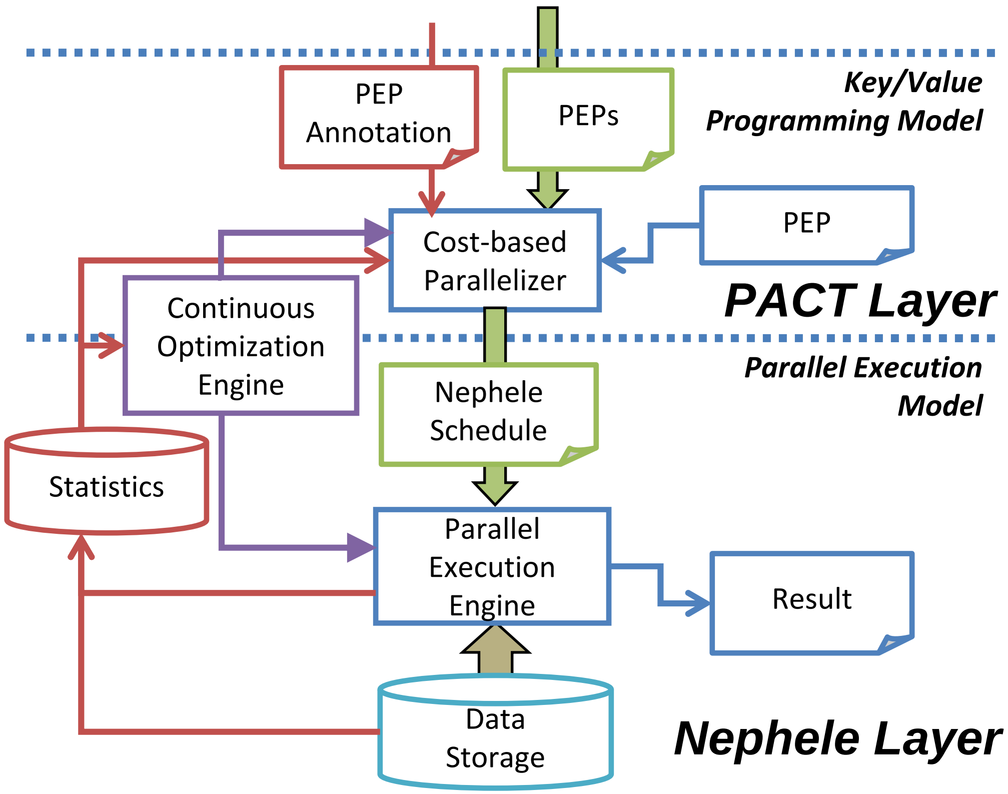
Architecture: PACT Layer

- Key Concepts
  - Massively parallel, fault-tolerant engine
  - **Declarative specification through parallelization contracts (PACTs)**



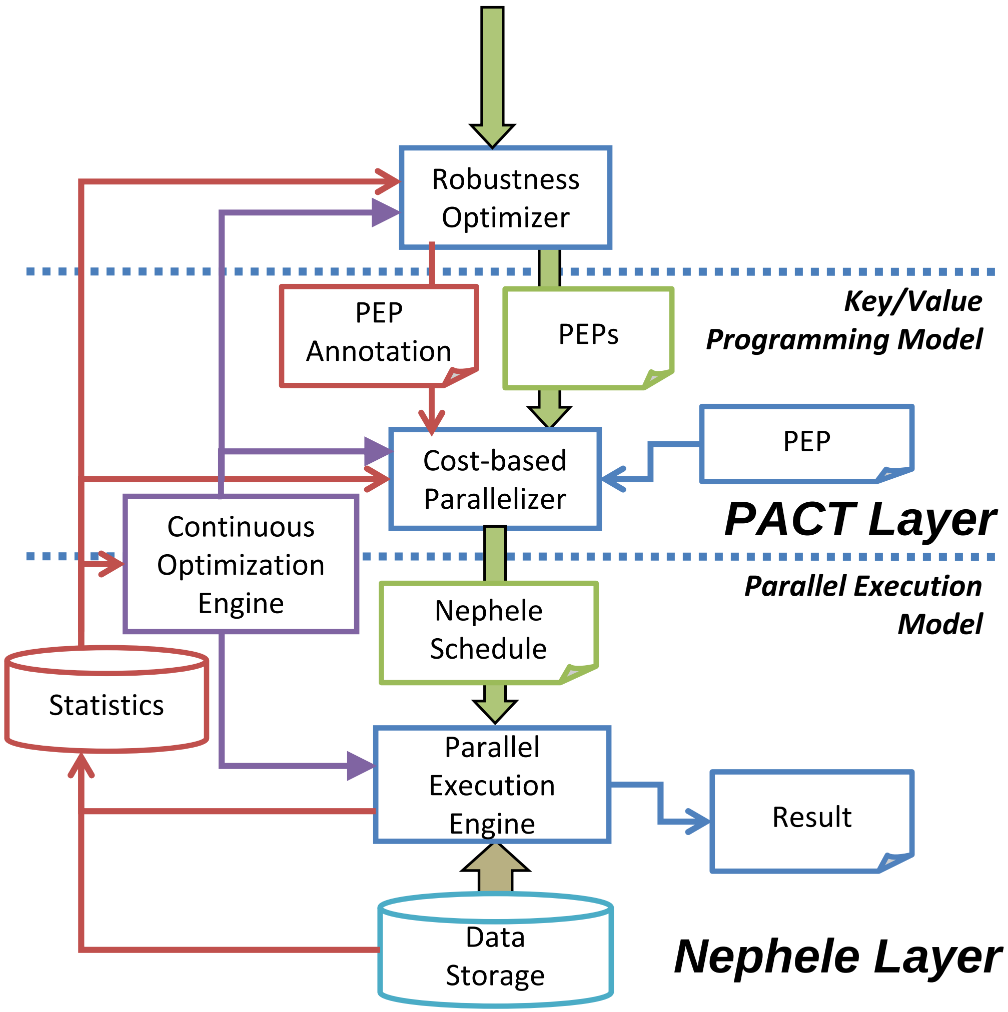
Architecture: Continuous Optimization

- Key Concepts
  - Massively parallel, fault-tolerant engine
  - Declarative specification through parallelization contracts (PACTs)
  - **Adaptive execution**



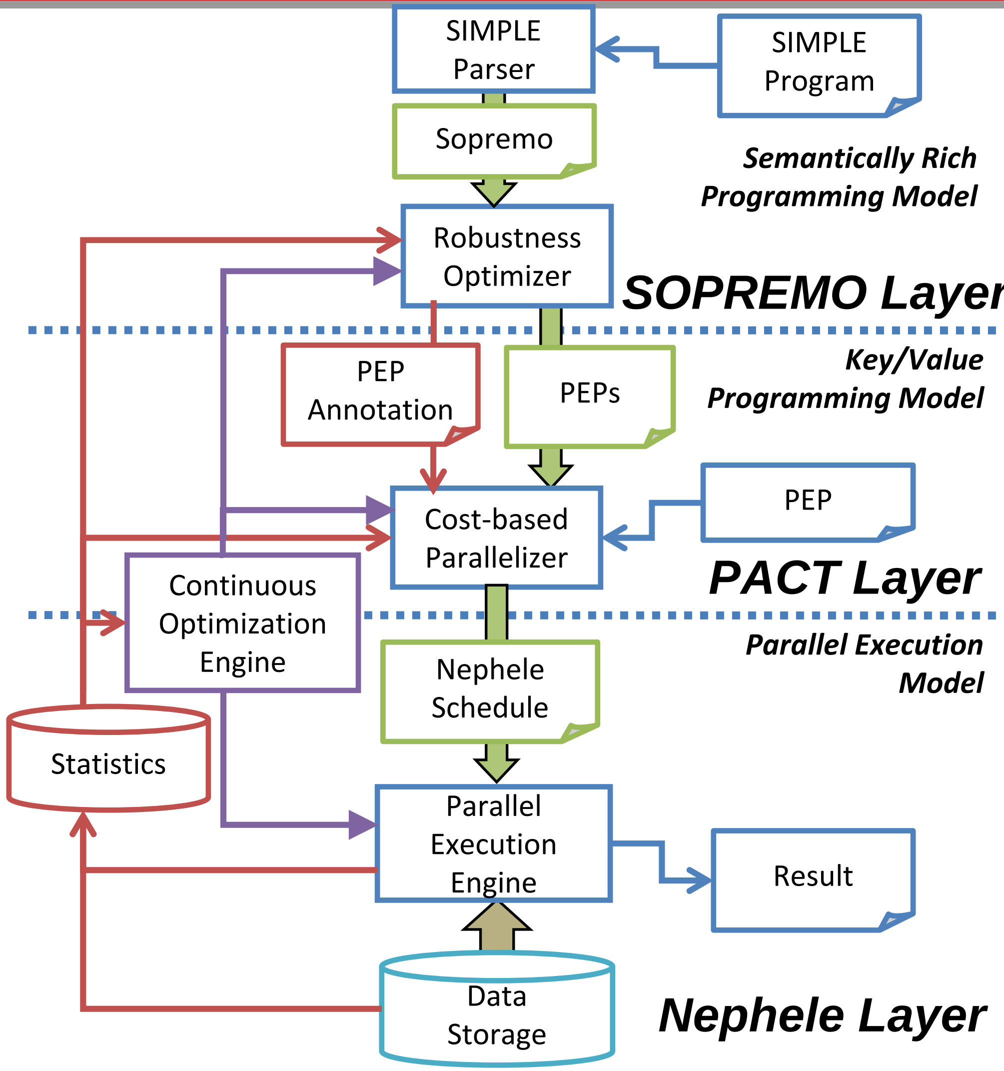
Architecture: Robustness

- Key Concepts
  - Massively parallel, fault-tolerant engine
  - Declarative specification through parallelization contracts (PACTs)
  - Adaptive execution
  - **Robust Optimization**



Architecture: SOPREMO Layer

- Key Concepts
  - Massively parallel, fault-tolerant engine
  - Declarative specification through parallelization contracts (PACTs)
  - Adaptive execution
  - Robust Optimization
  - **Semi-structured/text data model**
  - **Uncertainty**
  - **Declarative data flow programs with compute- and data intensive operations**
  - **Information extraction**
  - **Data cleansing**



What is a PACT?

- Second-order function that defines properties on the input and output data of its associated first-order function

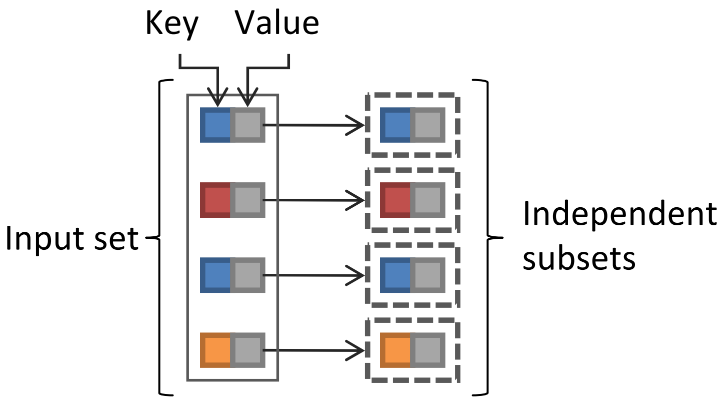
- Input Contract
  - Generates independently processable subsets of data
  - Generalization of map/reduce
  - Enforced by the system

- Output Contract
  - Describes properties of the output of the first-order function
  - Use is optional but enables certain optimizations
  - Guaranteed by the user

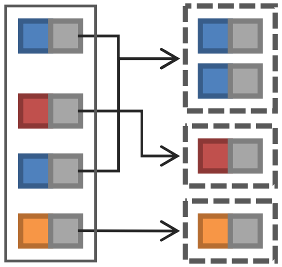
Map and reduce as PACTs

- Map and reduce are PACTs in our context

- Map
  - All pairs are independently processed

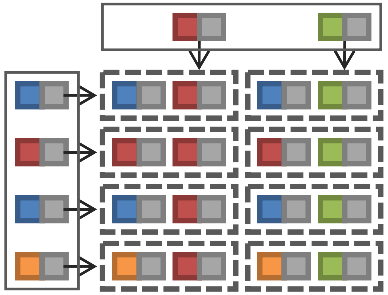


- Reduce
  - Pairs with identical key are grouped
  - Groups are independently processed

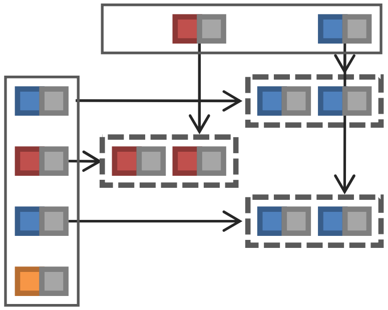


PACTs beyond Map and Reduce

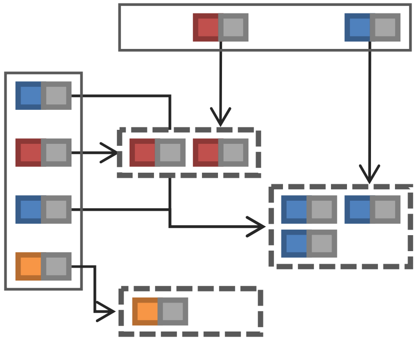
- Cross
  - Cartesian product of multiple inputs is built
  - All combinations are processed independently



- Match
  - Multiple inputs
  - All combinations of pairs with identical key over all inputs are built and processed independently
  - Contract resembles an equi-join on the key



- CoGroup
  - Pairs with identical key are grouped for each of multiple input
  - Groups of all inputs with identical key are processed together



Outline

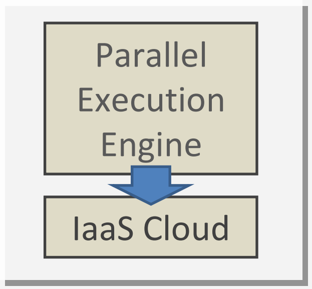
- Cloud Computing for Data Management
- Massive-parallel execution with Nephele
- Topology detection and streaming
- Conclusions

Research Question

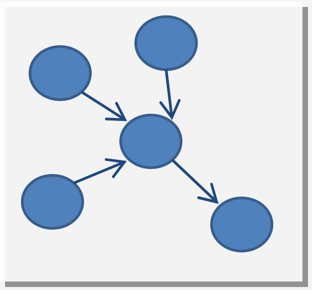
"How to improve the efficiency of massively parallel data processing on Infrastructure as a Service (IaaS) platforms"

- Opportunities: Elasticity
  - Scale-up/scale-down to respond to changes in the workload
  - Exploit resource heterogeneity to improve cost efficiency
- Challenges: Loss of control due to required virtualization
  - Shared infrastructure, loss of knowledge about I/O capacities
  - Network topology between machines is unknown

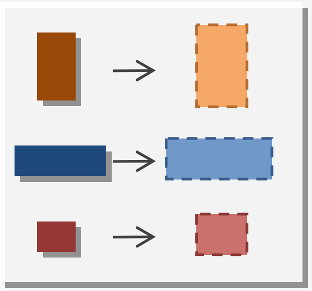
Requirements



- Shared resource management
  - Abandon assumption that execution engine “owns” nodes
  - Instead nodes are temporarily “leased”



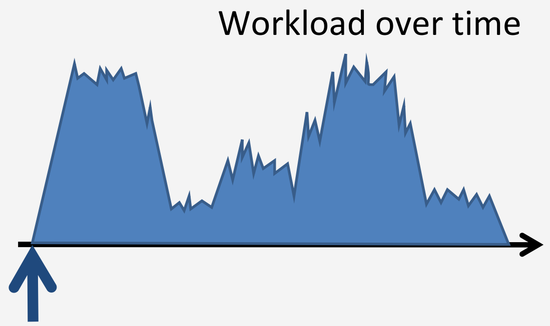
- Job must express tasks‘ data dependencies
  - Which task‘s input is required as which task‘s output
  - Required to safely terminate virtual machines

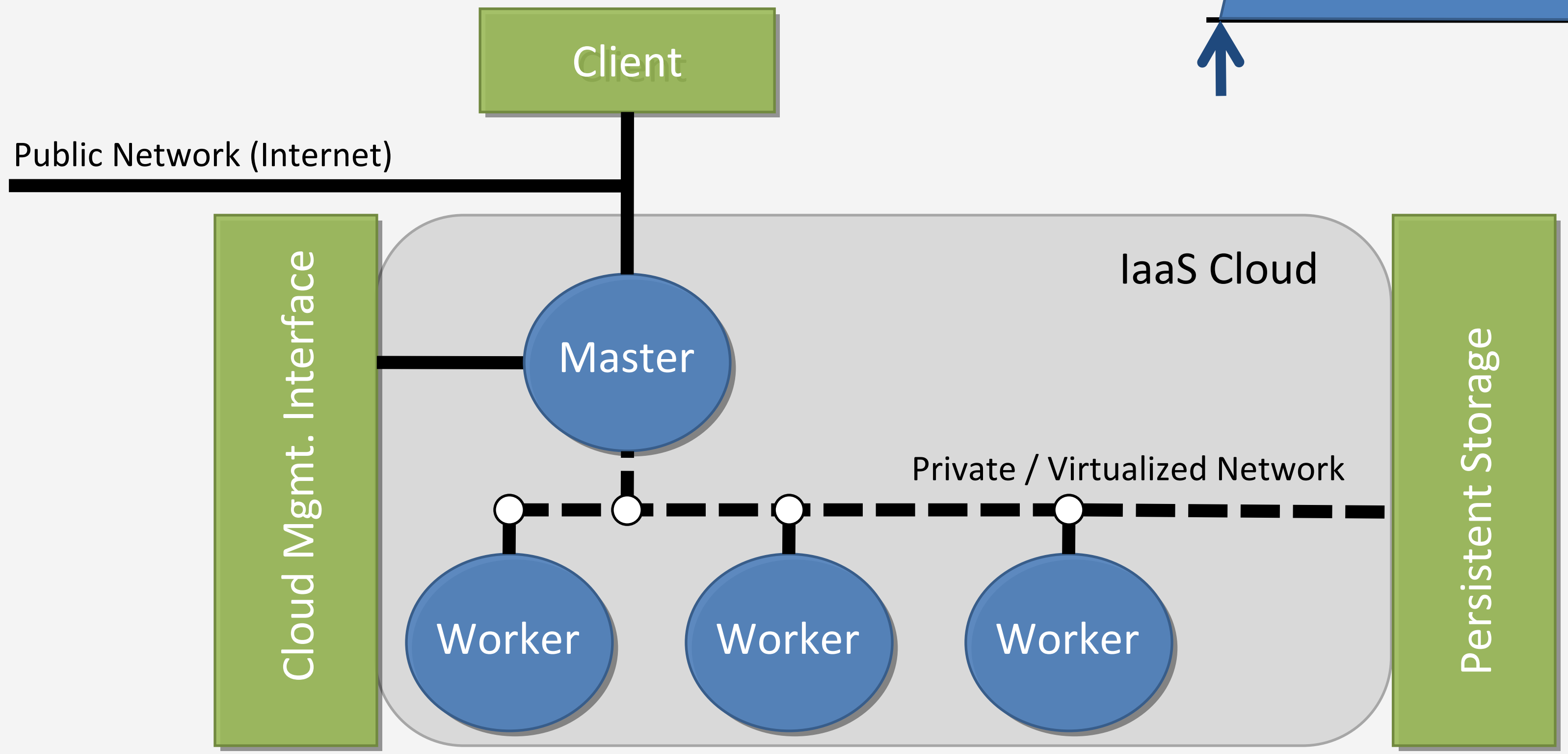


- Mapping between tasks and VM types
  - Which task shall run on which type of virtual machine?
  - Information could be provided by programmer

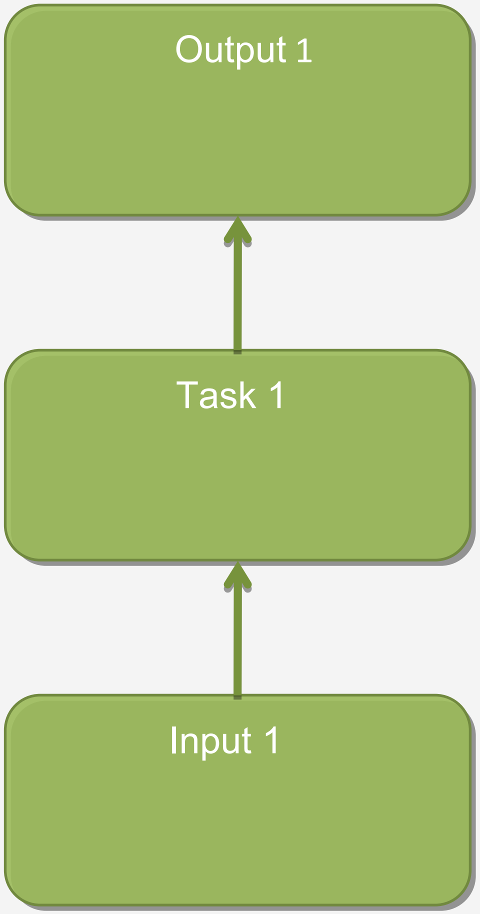
Research Prototype: Nephele

- Standard master worker pattern
- Workers can be allocated on demand



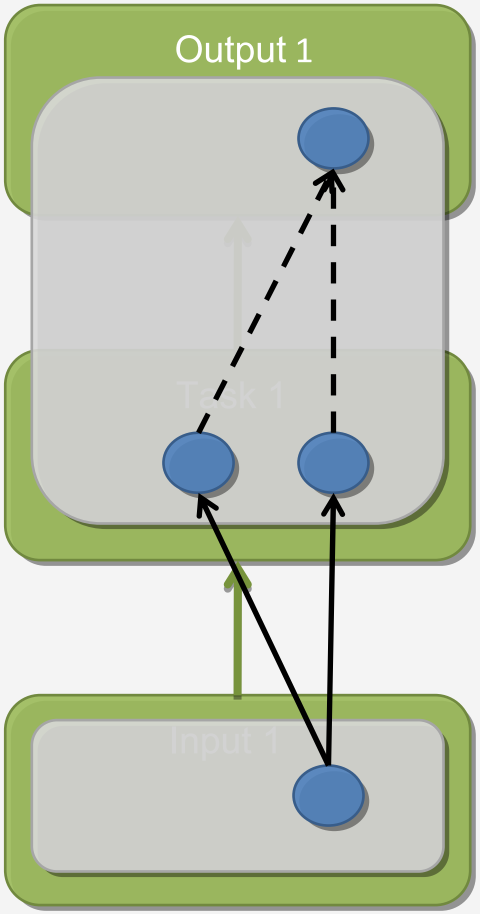


Nephele Job Description



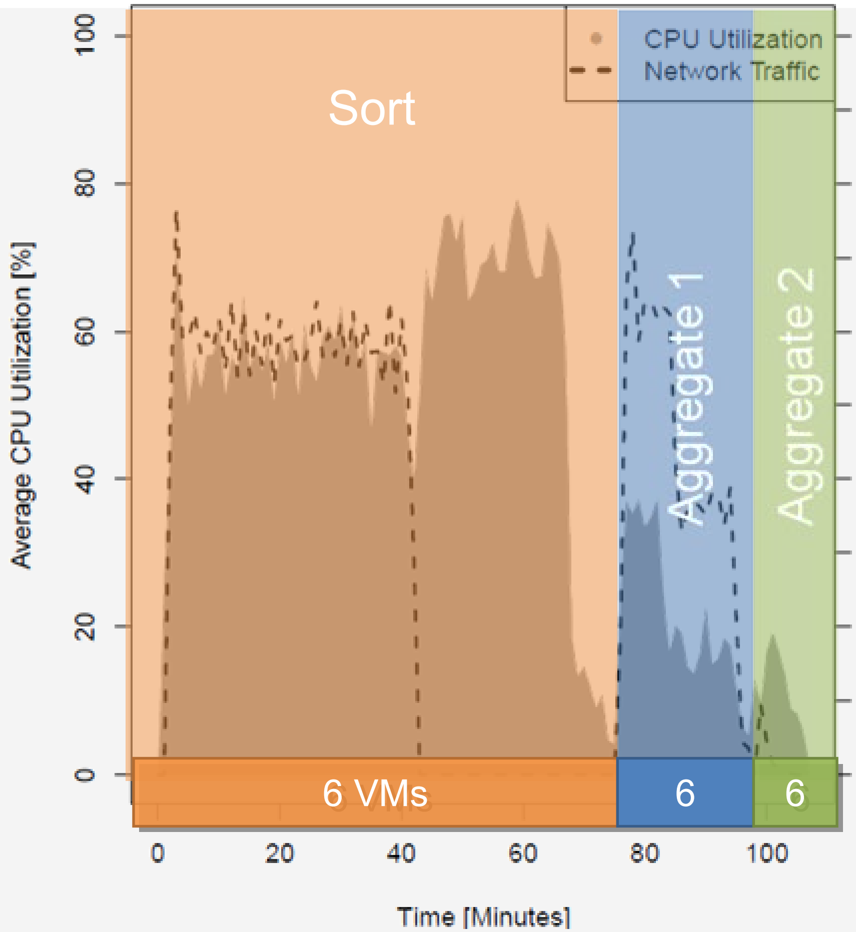
- Nephele job is represented as DAG
  - Vertices represent tasks
  - Edges denote communication channels
- Mandatory information for each vertex
  - Task program, (Input/output data location)
- Optional information for each vertex
  - Degree of parallelism
  - Degree of parallelism per node
  - Node type (#CPU cores, RAM…)
  - Channel types, …

Internal Scheduling Representation

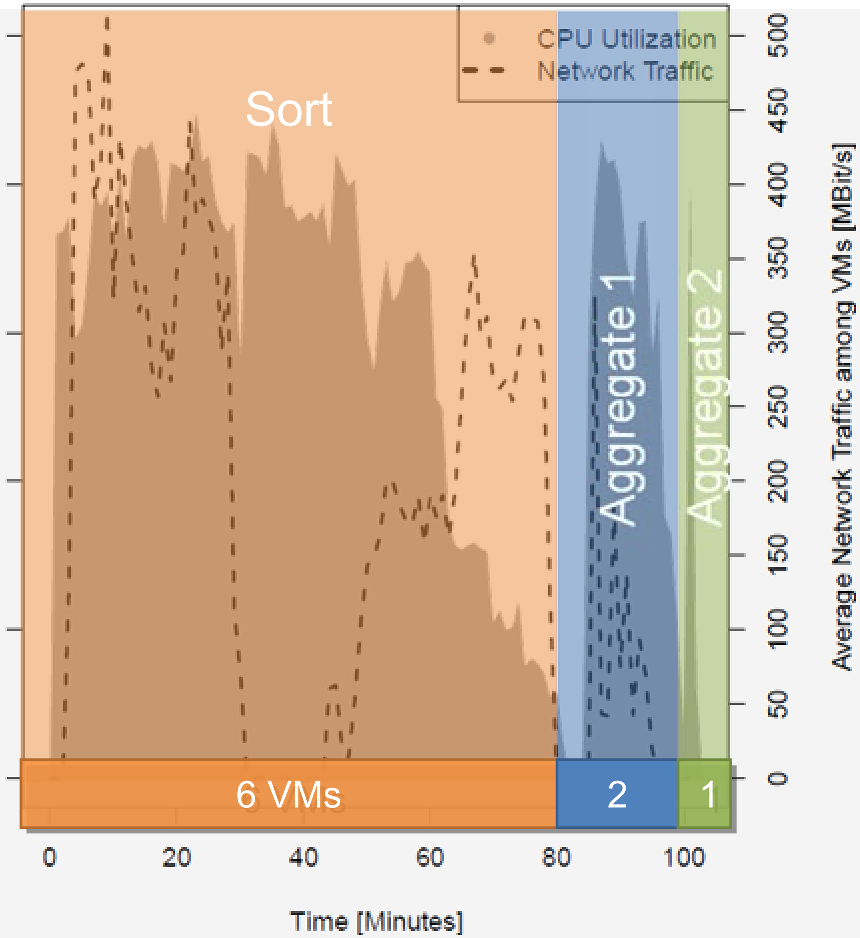


- Explicit parallelization
  - Individual degree of parallelization for each task
- Explicit assignment to VMs
- Communication channels
  - Network channels
  - In-memory channels
  - File channels

Experimental Evaluation



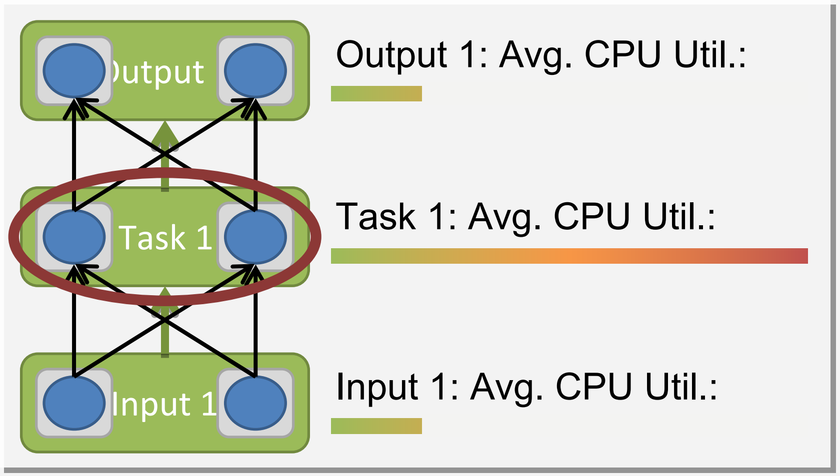
- MR jobs on Hadoop



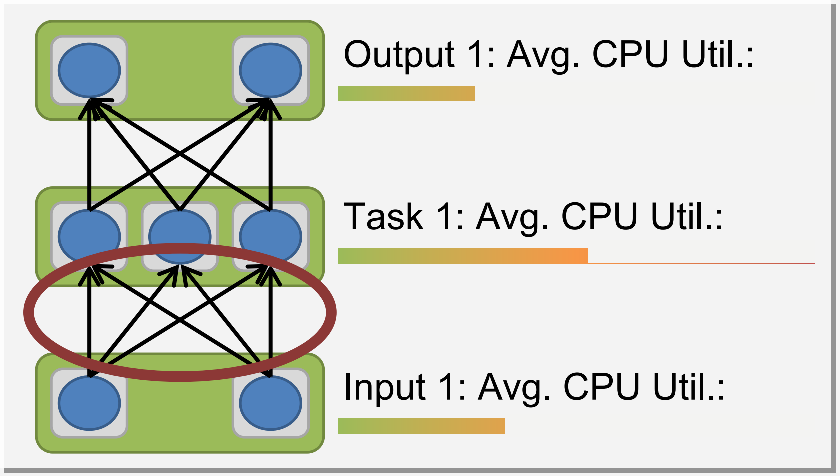
- MR jobs on Nephele

Challenges for Exploiting Elasticity

- Which degree of parallelization is suitable for which task?
  - Cloud philosophy: one core x 1000 hours = 1000 cores x one hour
  - Hard to anticipate for arbitrary user code, must be assessed online



CPU Bottlenecks



I/O Bottlenecks

Bottleneck Detection

- Profiling component runs on every worker node
- Profiling provides
  - *$pt(v_i)$*: % of time parallel instance *i* of vertex *v* used its given CPU time during last *t* seconds (seq. code, independence of par. instances)
  - *$st(e_j)$*: % of time parallel instance *j* of edge *e* was saturated during last *t* seconds (capacity contr. channels)
- Values of *$pt(v_i)$* and *$st(e_j)$* are propagated to master every *t* seconds

Bottleneck Detection Algorithm

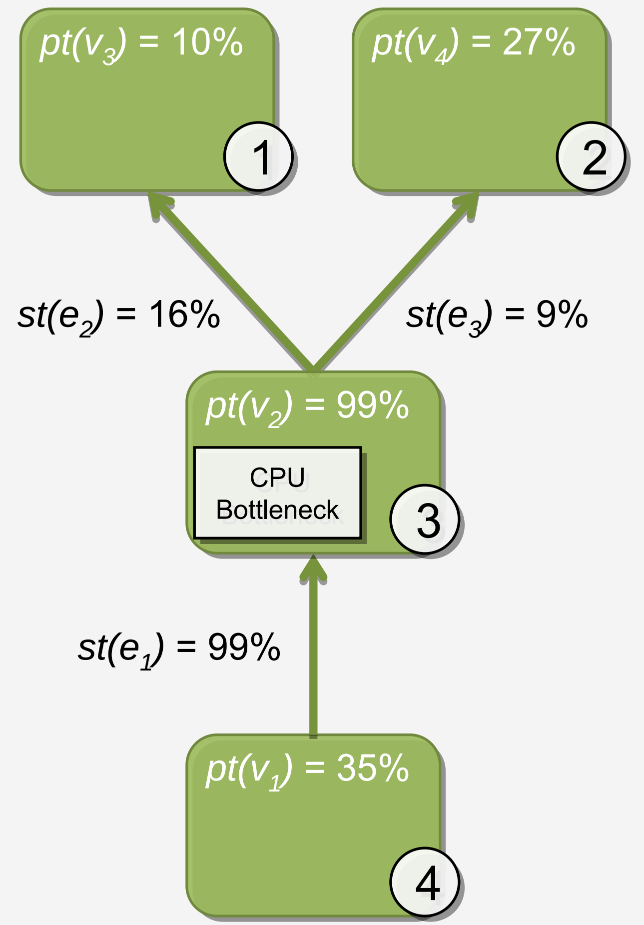
$L_{RTS} \leftarrow$ *ReverseTopologicalSort(G)*

for all v **in** L_{RTS} **do**
  v.isCpuBottleneck $\leftarrow$ *IsCPUBottleneck(v, G)*
end for

if $\nexists v \in L_{RTS}$: *v.isCPUBottleneck* **then**
  for all v **in** L_{RTS} **do**
    $E_v = \{(v,w) \mid w \in V_G \Box (v,w) \in E_G\}$
    for all $e \in E_v$ **do**
      e.isIOBottleneck $\leftarrow$ *IsIOBottleneck(e, G)*
    end for
  end for
end if

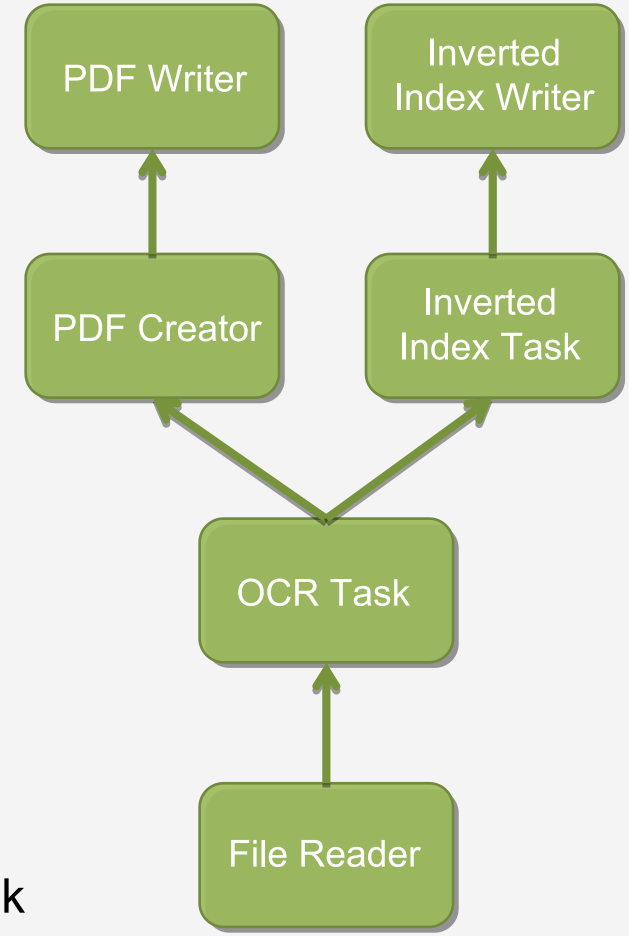
Criteria CPU bottleneck:
- $pt(v) > \alpha$ (α = 90%)
- No successor vertex of *v* is CPU bottleneck

Criteria I/O bottleneck:
- $st(e) > \beta$ (β = 90%)
- No successor edge of *e* is I/O bottleneck

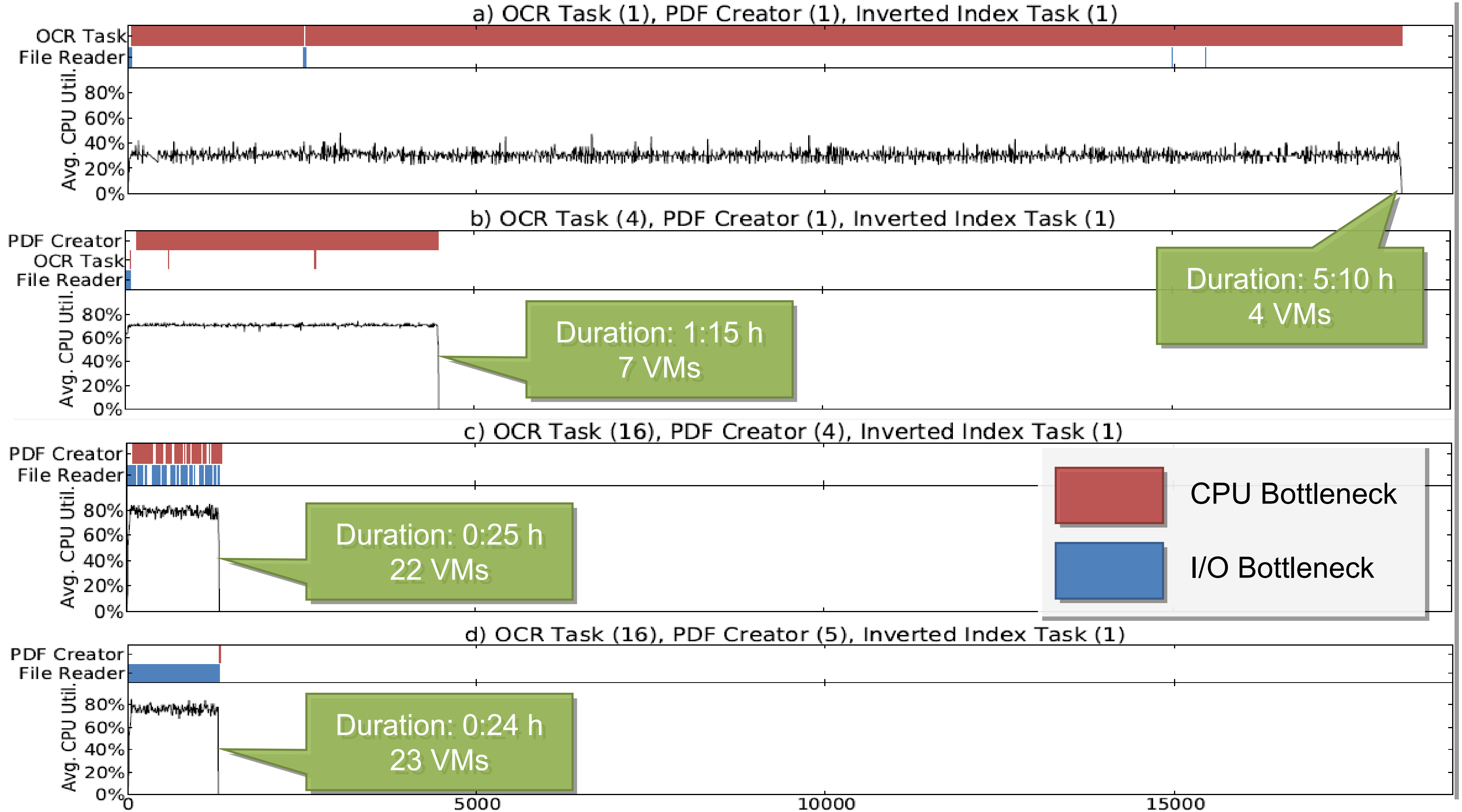


Evaluation (1/2)

- Evaluation job
  - Conversion of article DB
  - 40 GB of bitmap images to PDF
- Properties of job
  - Different computational complexities of tasks
  - Each parallel instance runs on separate VM (with 1 CPU core)
  - Input data reside on external storage
- Goal of evaluation
  - Find ideal degree of parallelization for each task



Evaluation (2/2)

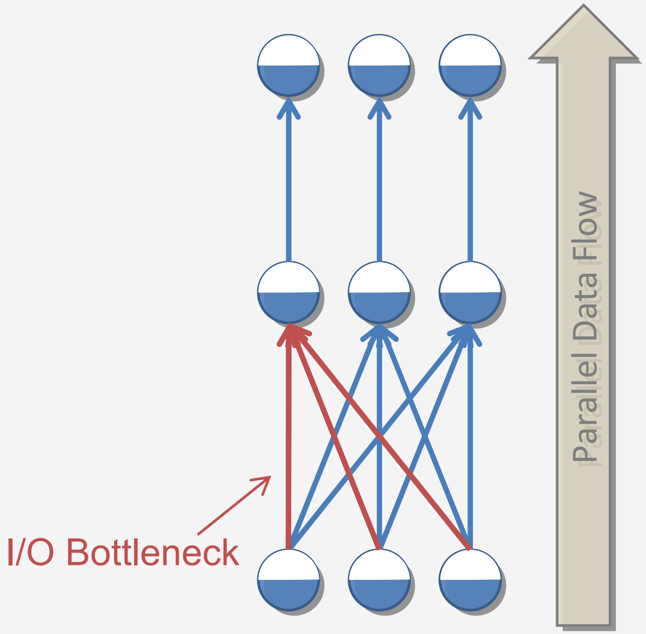


Outline

- Cloud Computing for Data Management
- Massive-parallel execution with Nephele
- Topology detection and adaptive compression
- Conclusions

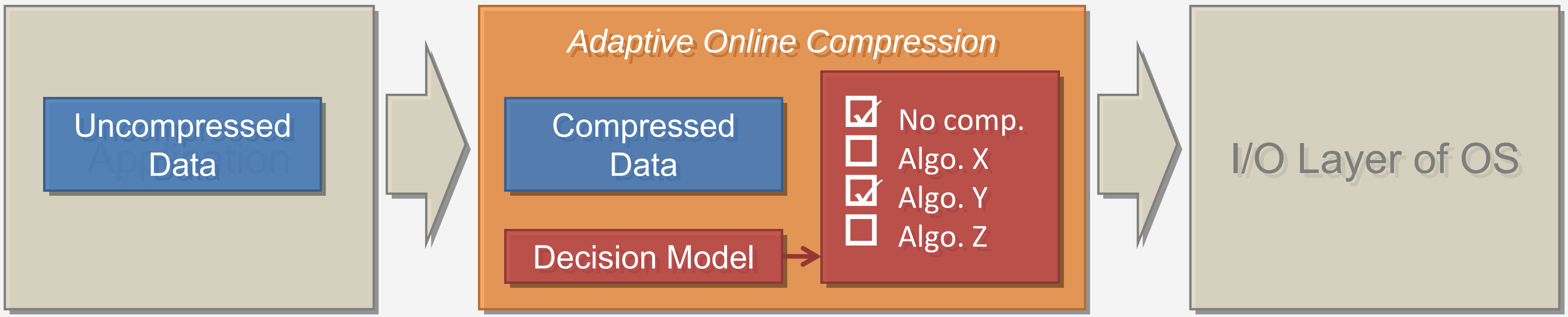
Motivation

- The network is a scarce resource
  - Used for communication among nodes
  - Used by distributed file system
  - Possibly used by other virtual machines
- Network performance hard to predict
  - Available throughput may change over time
  - Can lead to I/O bottlenecks starvation
- Idea: Handle varying I/O performance on application layer
  - Adaptive compression
  - Topology detection

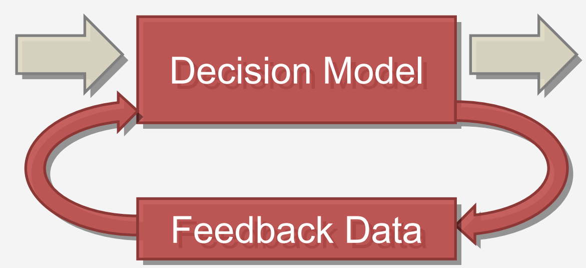


Adaptive Online Compression

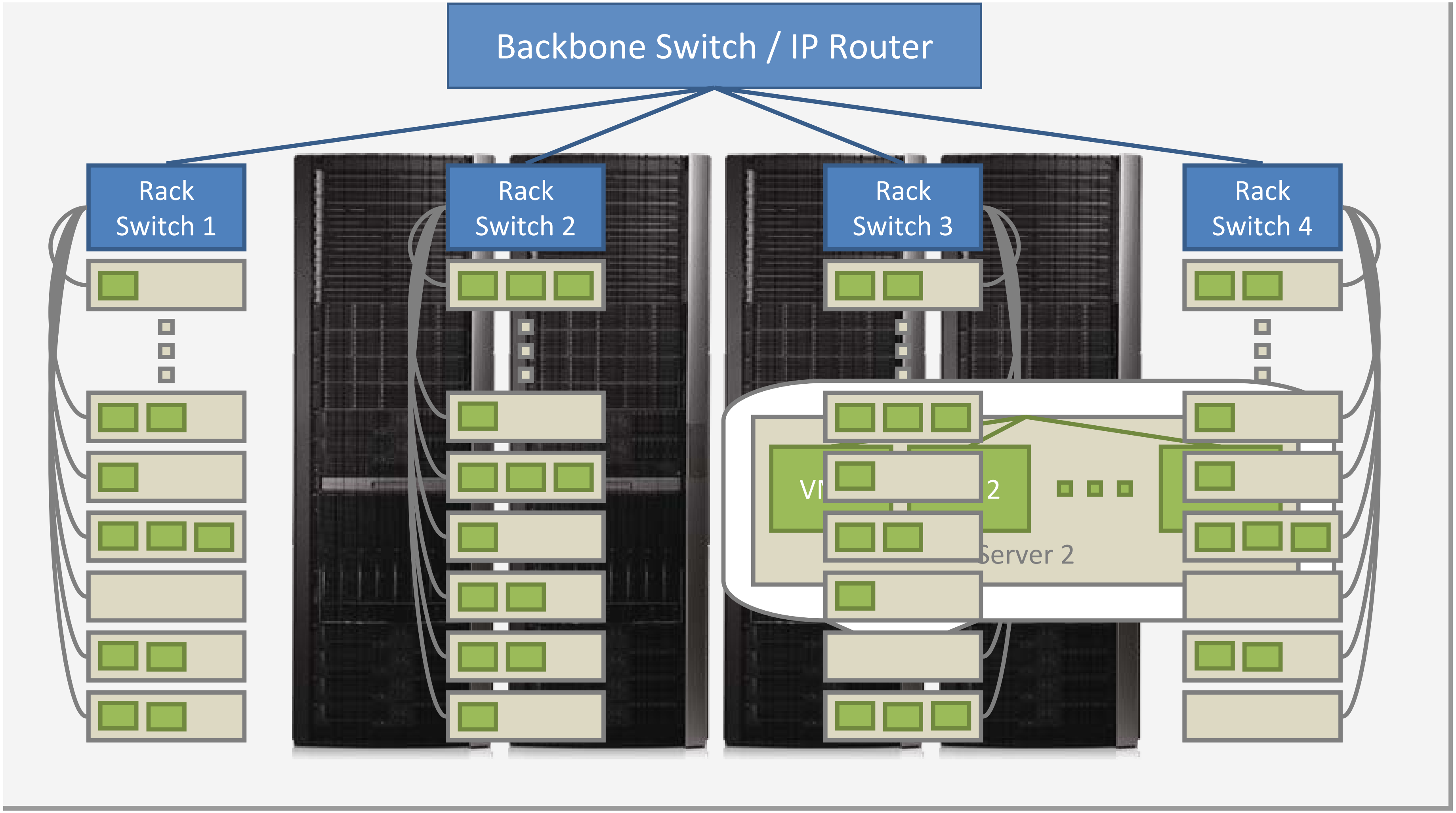
- Selection of different compression algorithms
  - Each algorithm has different time/size ratio



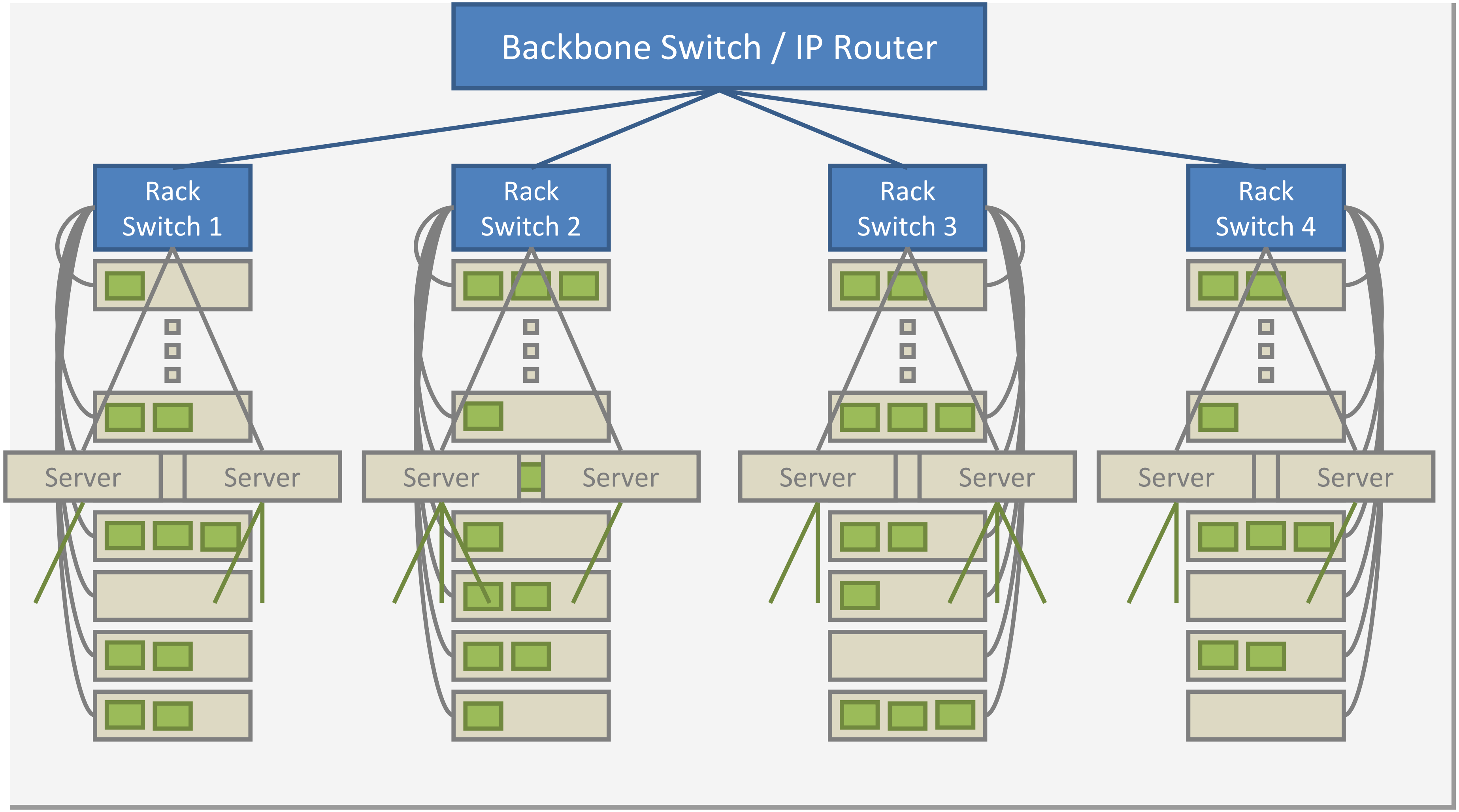
- Calibration of decision model during data transfer
  - Try out different compression levels
  - Learn from previous compression decisions
  - Reward good decisions, penalize bad ones



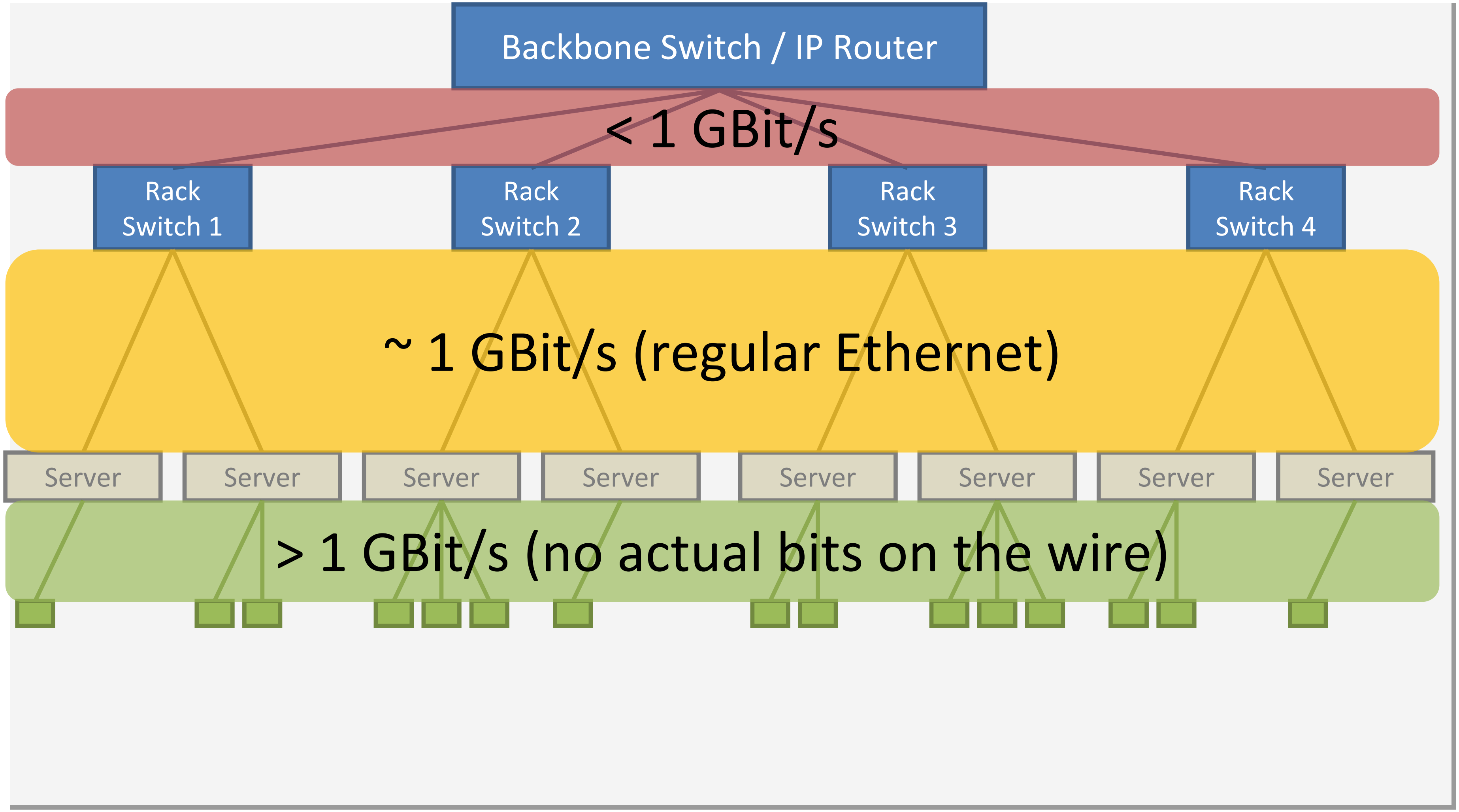
Detecting network topology



Detecting network topology



Detecting network topology

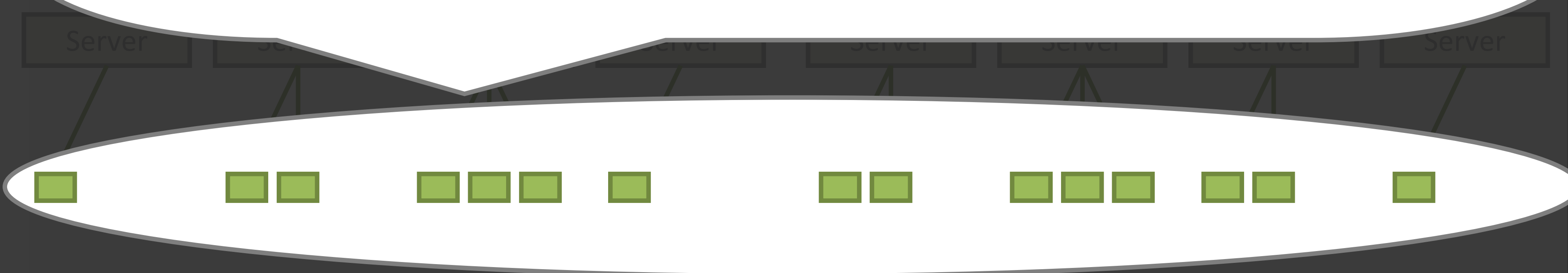


Detecting network topology

- Cloud costumer's perspective:
  - IP addresses to VMs only ⇒ Underlying network topology is not revealed
  - Data locality cannot be exploited inside application

```
warneke@hadoop-dev:~$ euca-describe-instances
RESERVATION     r-310C06F3      marrus  default
INSTANCE        i-348C06AC      emi-AE291B0F    192.168.198.14  192.168.198.14  running mykey   2
INSTANCE        i-3A1E062D      emi-AE291B0F    192.168.198.13  192.168.198.13  running mykey   0
INSTANCE        i-46BC0853      emi-AE291B0F    192.168.198.12  192.168.198.12  running mykey   1
```

- Can we infer the physical network topology from the VMs?



Topology Inference (TI) from End Nodes

- Rely on assistance of internal network nodes
  - Use ICMP, traceroute-like tools

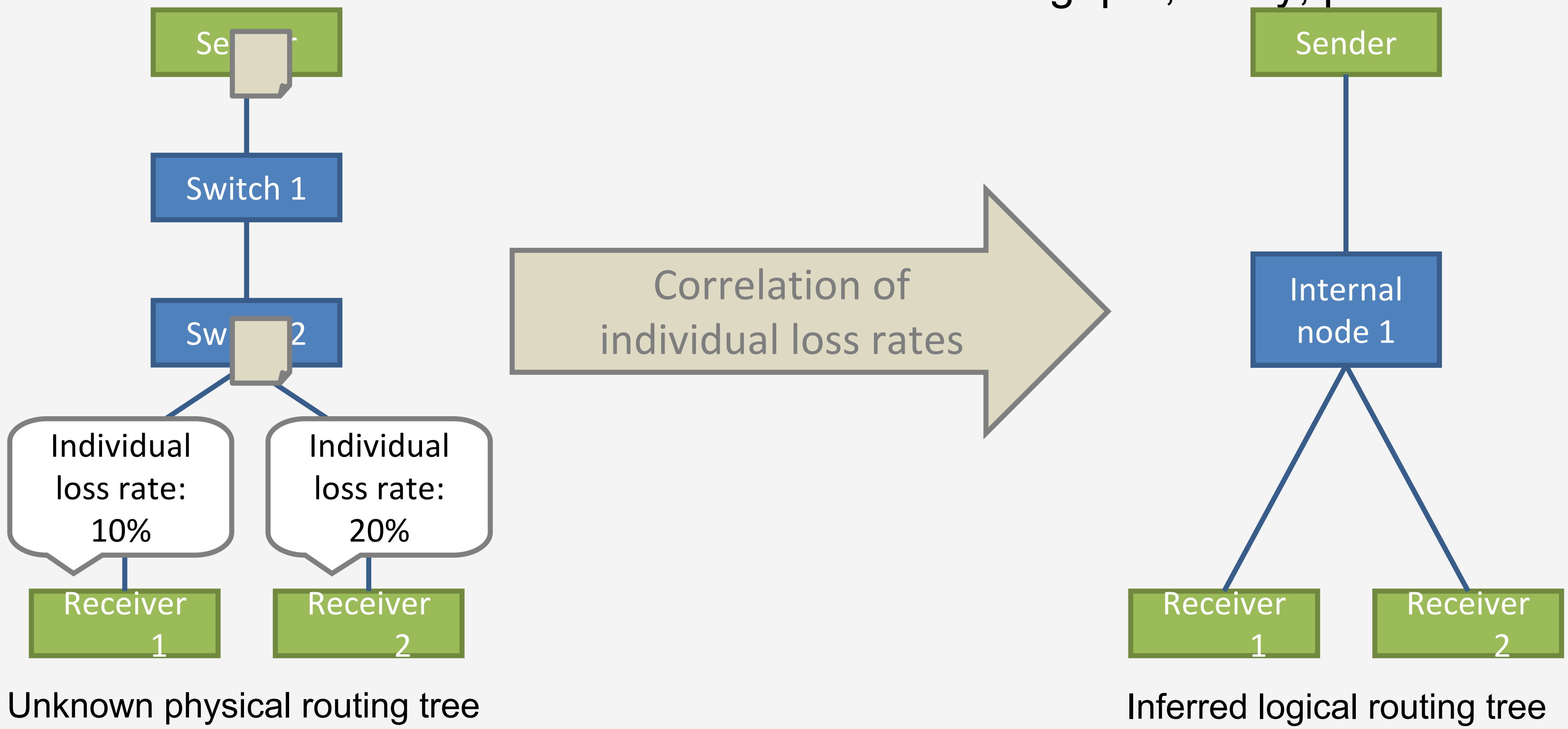
Benefits	Challenges
✓ Simple	× Unable to detect switches/bridges
✓ Robust for IP-level topologies	× Anonymous routers

- Do not rely on assistance of internal network nodes
  - Observe network behavior from end nodes only
  - Use observations to infer existence of internal network nodes

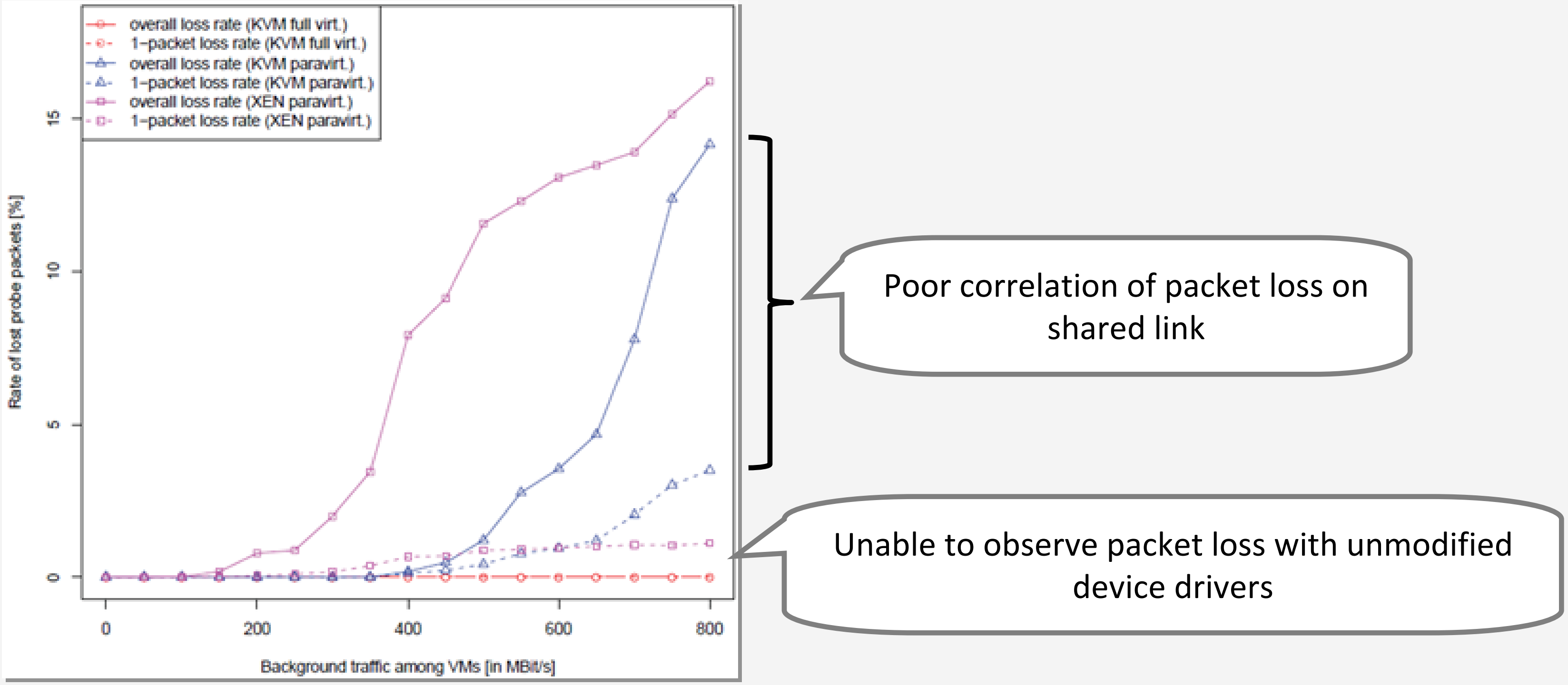
Benefits	Challenges
✓ > 10 years research history for WANs	× No research for data center networks
✓ Potentially identifies switches/bridges	× Impact of virtualization unknown

TI based on End-to-End Measurements

- One sender node, two or more receiver nodes
  - Connected through unknown, tree-like network
  - Sender sends probe packets to receivers
  - Receivers observe link characteristics like throughput, delay, packet loss

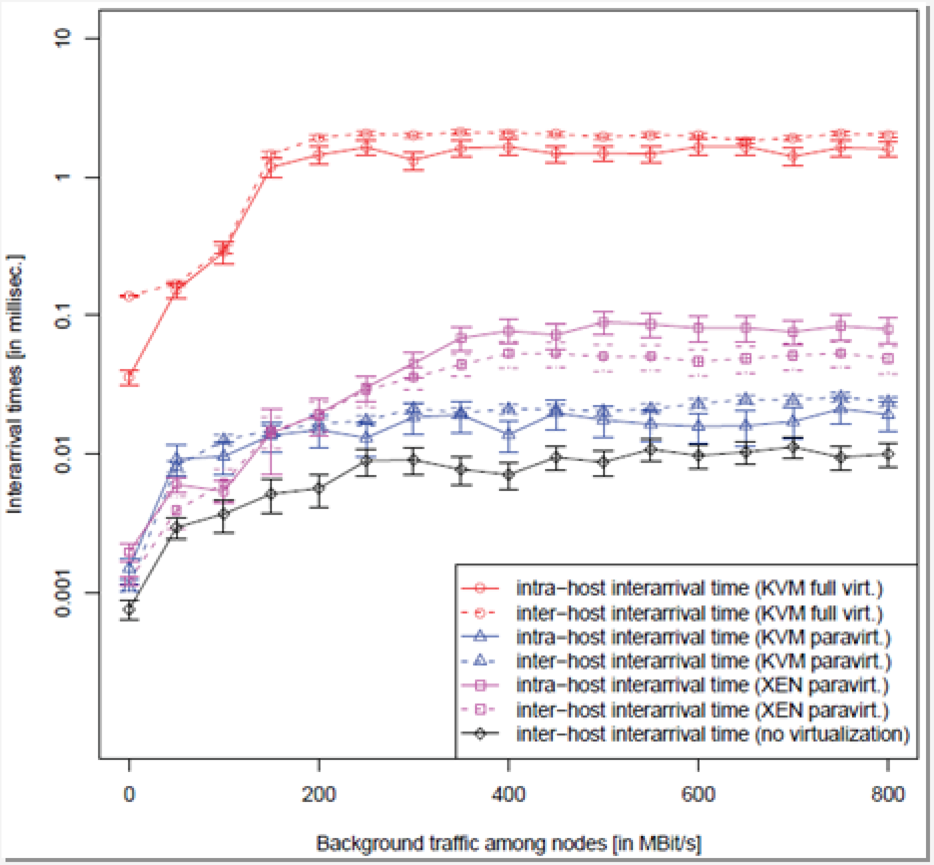


Link Characteristic Packet Loss



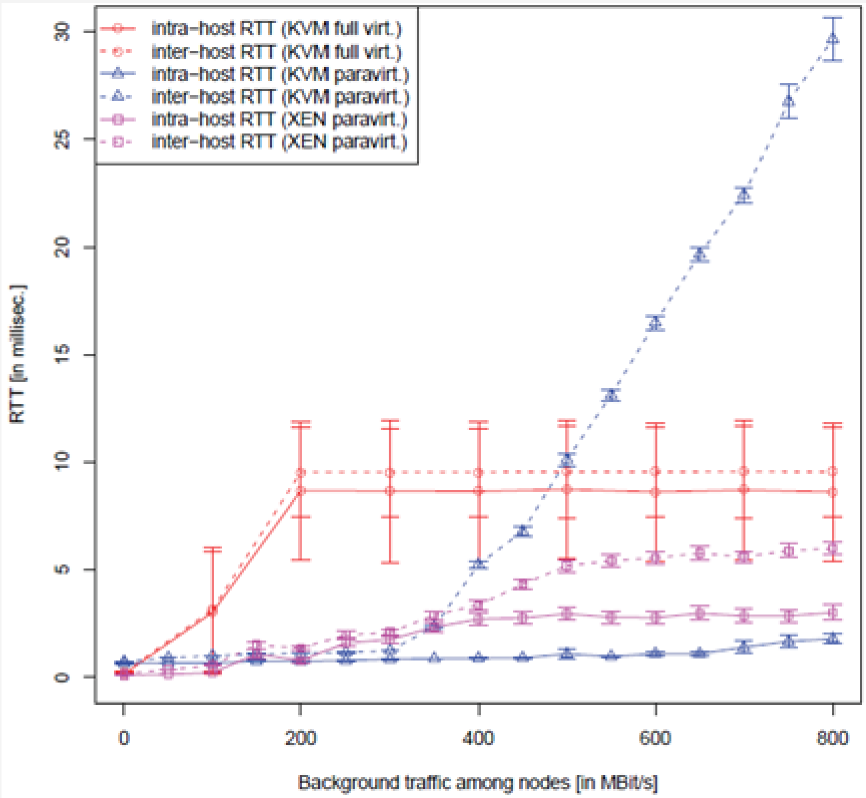
- Packet loss hard to observe due to high throughput links
- Virtualization destroys packet correlation on shared link

Link Characteristic Delay



- Poor delay correlation for KVM with unmodified device drivers
- Modest increase of interarrival times for both KVM and XEN (paravirtualization)

Link Characteristic Delay (RTT)



Statistically significant gap between intra- and intra-host RTT for XEN paravirt.

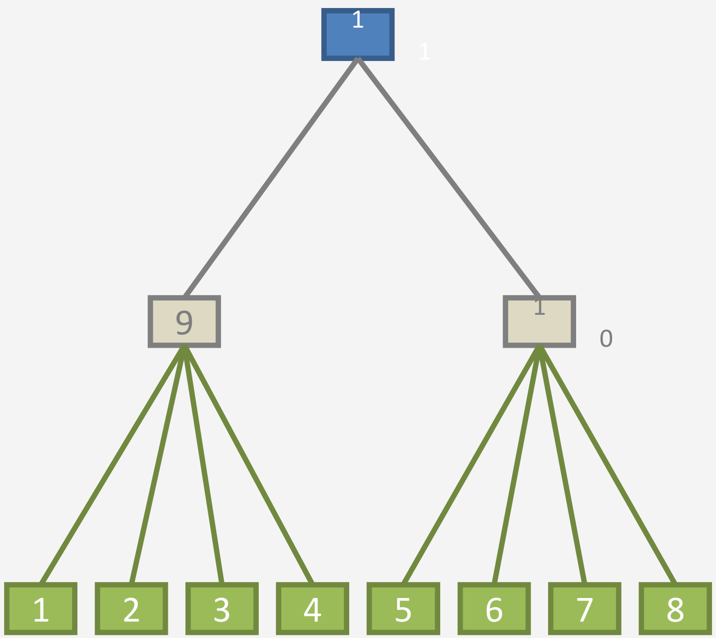
High variance of RTTs for KVM full virt.

Statistically significant gap between intra- and intra-host RTT for KVM paravirt.

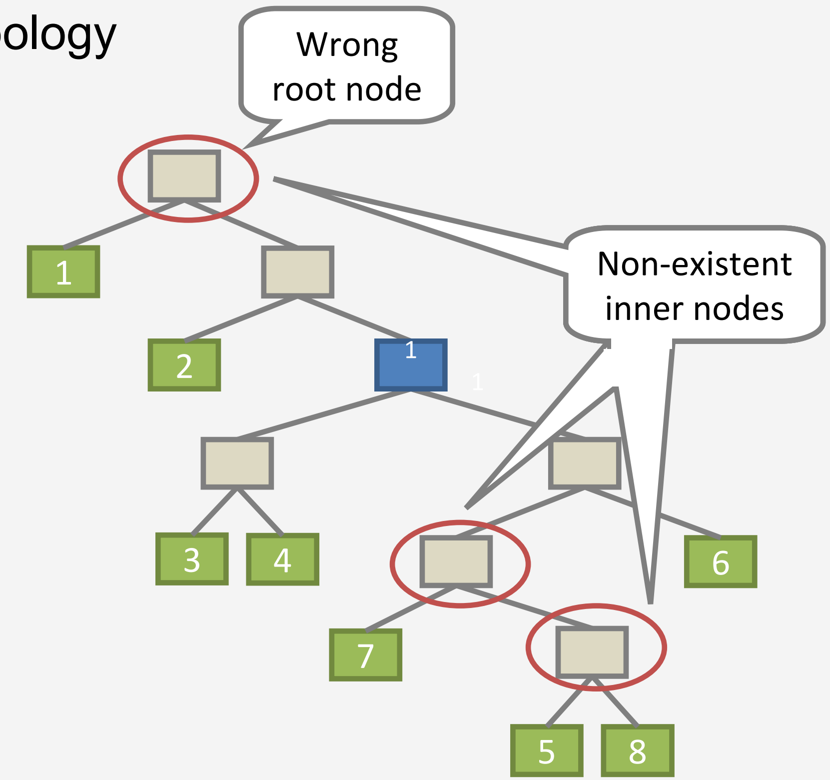
- RTT can be used to detect co-located VMs with paravirt.

Inferred Tree is always Binary

- Binary trees fit measured data most closely
  - Highest degree of freedom
  - „Overfitted“ version of actual network topology



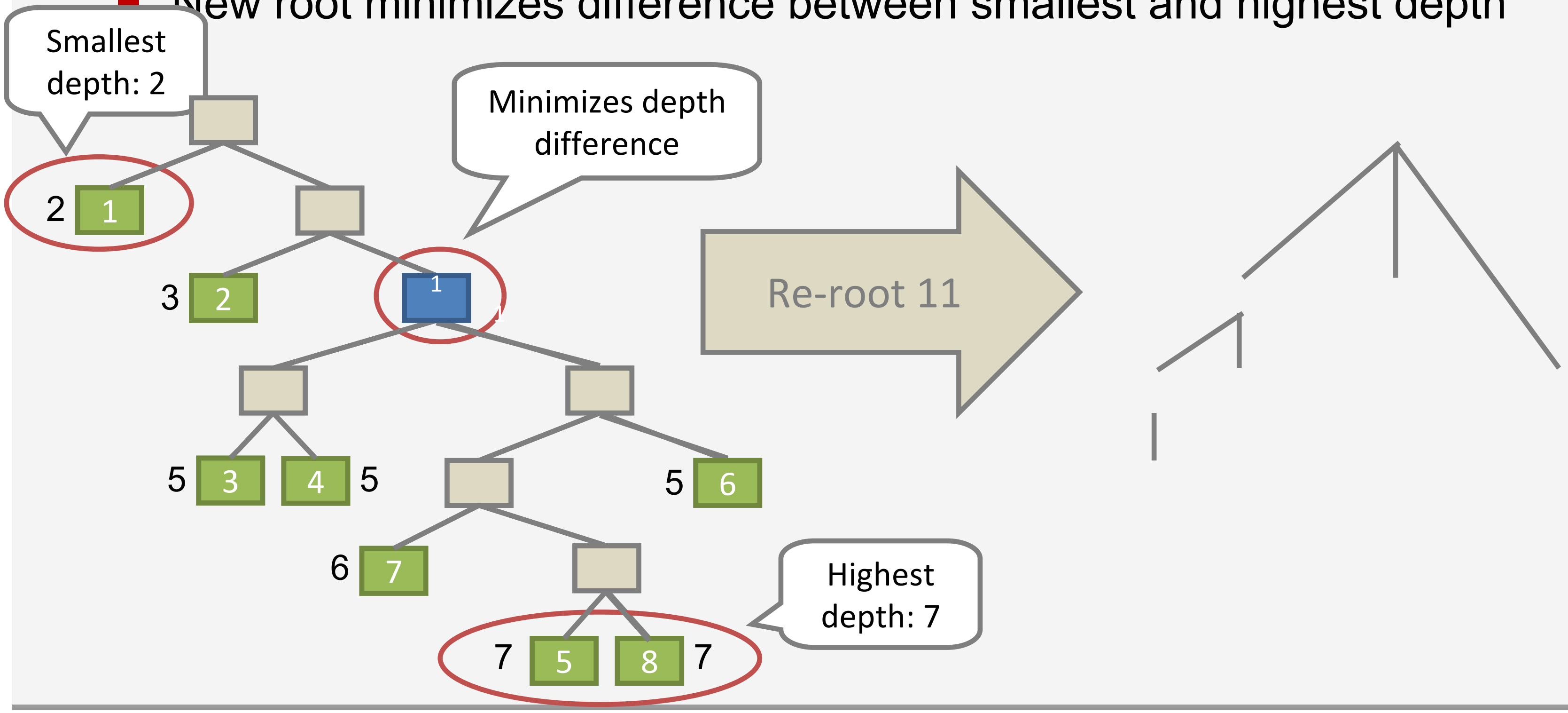
Physical routing tree



Inferred logic routing tree

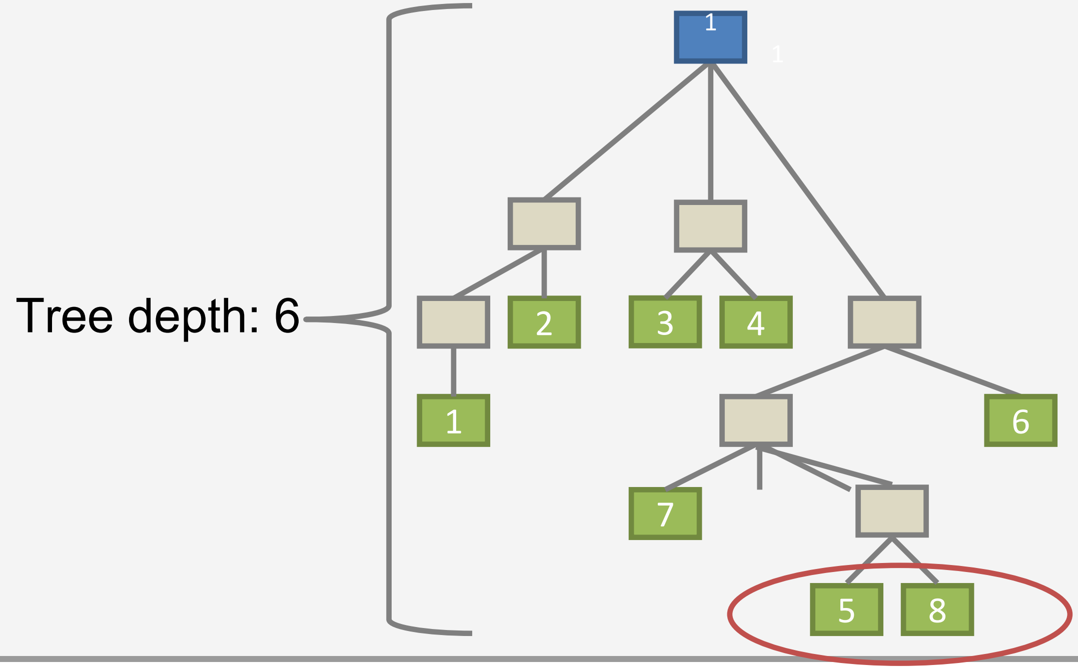
Re-Rooting the Inferred Tree

- Remember: Data center networks have regular structure
- Idea:
  - Determine depth of each leaf node
  - New root minimizes difference between smallest and highest depth



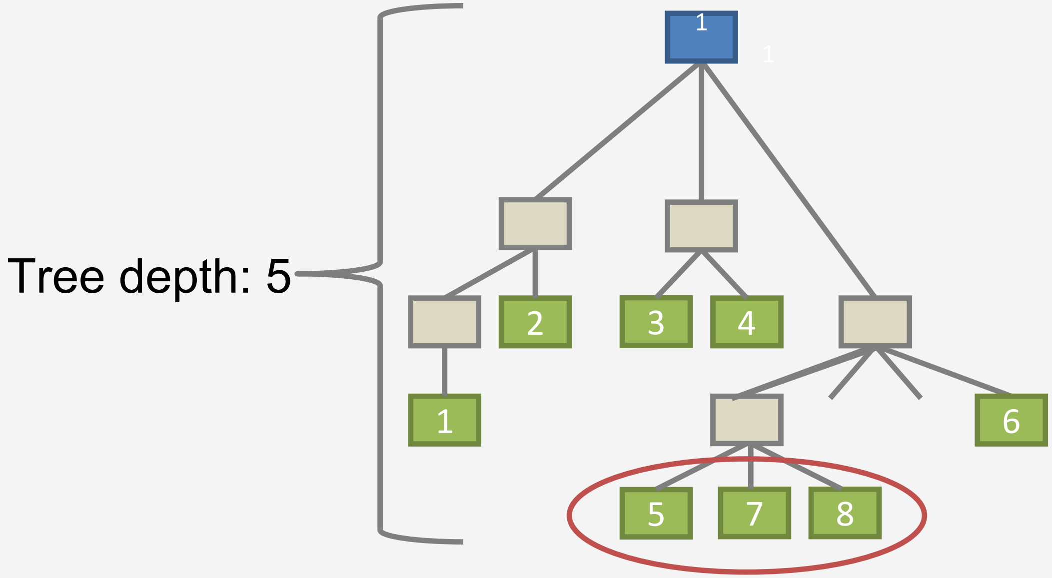
Limiting Depth of Inferred Tree

- After Re-rooting, depth of the inferred tree is reduced
  - Assumption: Tree depth greater than *d* is unlikely to occur in data center
- Idea:
  - Until tree depth ≤ *d*, identify leaf node with highest depth
  - Merge parent and parent's parent



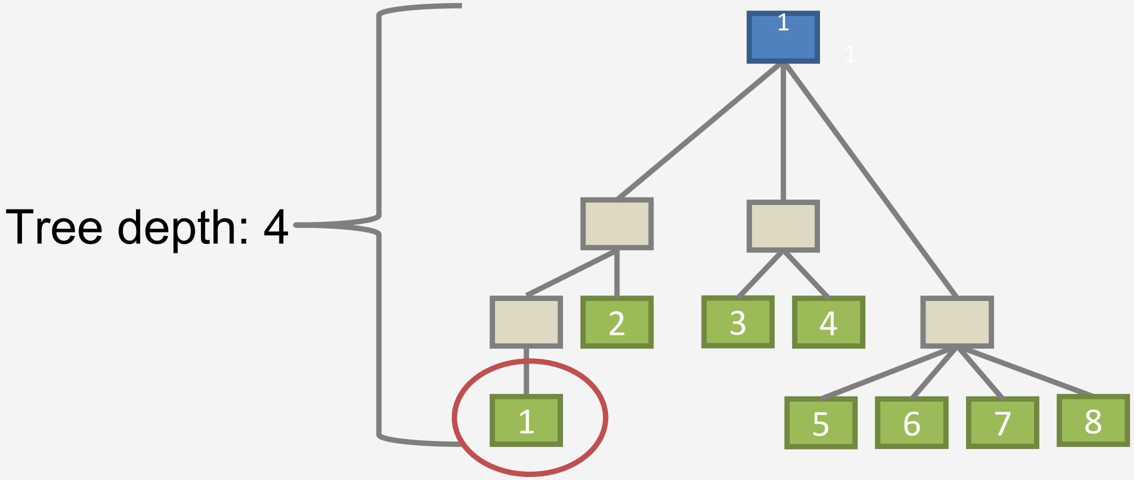
Limiting Depth of Inferred Tree

- After Re-rooting, depth of the inferred tree is reduced
  - Assumption: Tree depth greater than *d* is unlikely to occur in data center
- Idea:
  - Until tree depth ≤ *d*, identify leaf node with highest depth
  - Merge parent and parent's parent



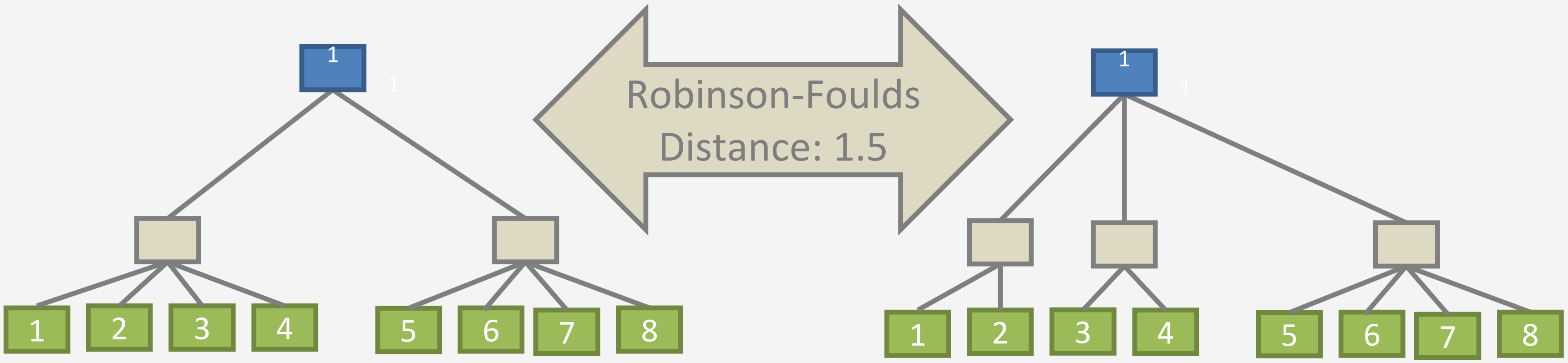
Limiting Depth of Inferred Tree

- After Re-rooting, depth of the inferred tree is reduced
  - Assumption: Tree depth greater than *d* is unlikely to occur in data center
- Idea:
  - Until tree depth ≤ *d*, identify leaf node with highest depth
  - Merge parent and parent's parent



Limiting Depth of Inferred Tree

- After Re-rooting, depth of the inferred tree is reduced
  - Assumption: Tree depth greater than *d* is unlikely to occur in data center
- Idea:
  - Until tree depth ≤ *d*, identify leaf node with highest depth
  - Merge parent and parent's parent

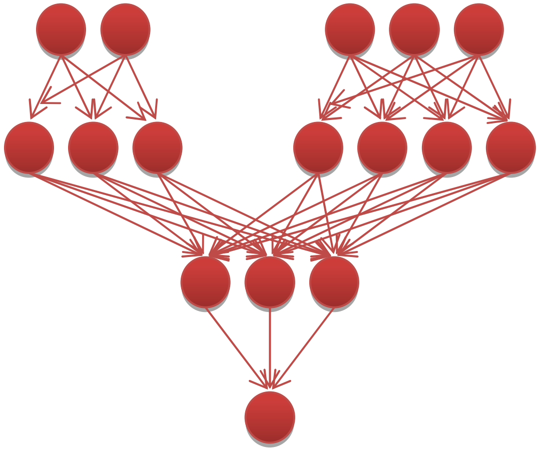


Physical routing tree

Inferred logic routing tree

Current Work: Streaming

- Nephele and PACTs currently focus on batch-job workloads
  - Usual goal: „minimize time-to-solution"
  - Translates to „maximize throughput"



- What about streaming workloads?
  - Possible with Nephele, but (as of now) not PACTs
  - May have different goals
    - Meet pipeline latency and throughput requirements
    - Minimize pipeline latency, don't care about throughput
    - Max/Min other custom metrics

Conclusion

- Parallel data processing on clouds is promising research area
  - Elasticity/cost model provides new use cases
- Future work
  - Streaming and profile comparisons
  - CloudNets – move part of the computation into the networks
- Plenty of opportunities for future work
  - Currently 20+ developers, Apache License
  - Check www.stratosphere.eu for downloads, tutorials

Thank you

www.cit.tu-berlin.de