

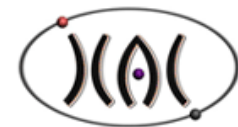


Predictable Time-Sharing for DryadLINQ Cluster

Sang-Min Park and Marty Humphrey

Dept. of Computer Science
University of Virginia





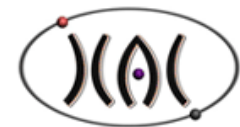
DryadLINQ

- What is DryadLINQ?
 - LINQ: Data processing language and run-time in .NET
 - DryadLINQ: LINQ impl. for cluster computers
 - Similar to popular MapReduce/Hadoop

- Sample LINQ program

```
IEnumerable<string> GetTopKWord(IEnumerable<string> input, int k)
{
    return input
        .SelectMany(x => x.Split(" "))           // line -> words
        .GroupBy(x => x)                       // word -> (word, #(word))
        .OrderByDescending(x => x.Count())     // sort by word's occurrence
        .Take(k)                               // take top k
        .Select(x => x.Key);                   // transform to string type
}
```





DryadLINQ



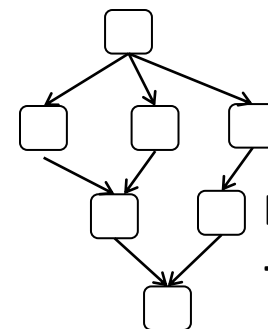
```

IEnumerable<string> GetTopKWord( ... )
{
    return input
    .SelectMany(x => x.Split(' '))
    .GroupBy(x => x)
    .OrderByDescending(x => x.Count())
    .Take(k)
    .Select(x => x.Key);
}

```



**LinqToDryad
Compiler**



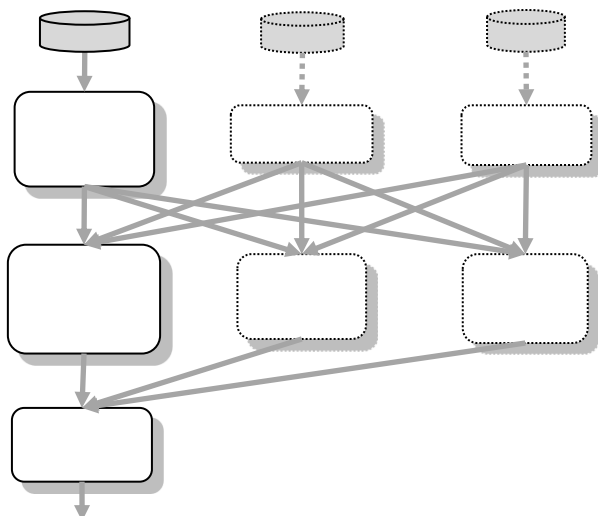
**DAG task plan
+ compiled DLL**



**Cluster
head
node/scheduler**



**Dryad
Run-time**



**DryadLINQ program
(client desktop)**





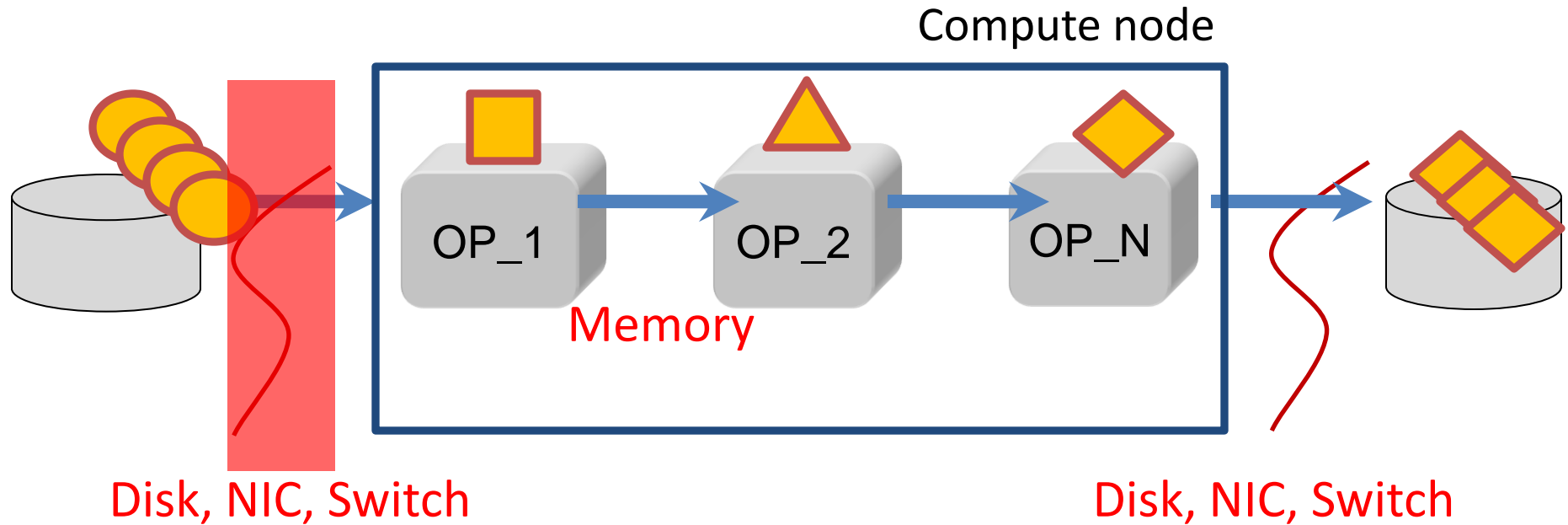
DryadLINQ and MapReduce

- Commonalities
 - Inspired by functional programming
Map(fx)-Reduce(gx)
Where(fx)-OrderBy(gx)
 - Streamlined data processing
 - Master-worker computation model
- Differences
 - Program structure (DAG vs. Map-Reduce)
 - DryadLINQ *compiles* user's program
 - Supported operations (MapReduce < DryadLINQ)
 - Open source (Hadoop) vs. proprietary



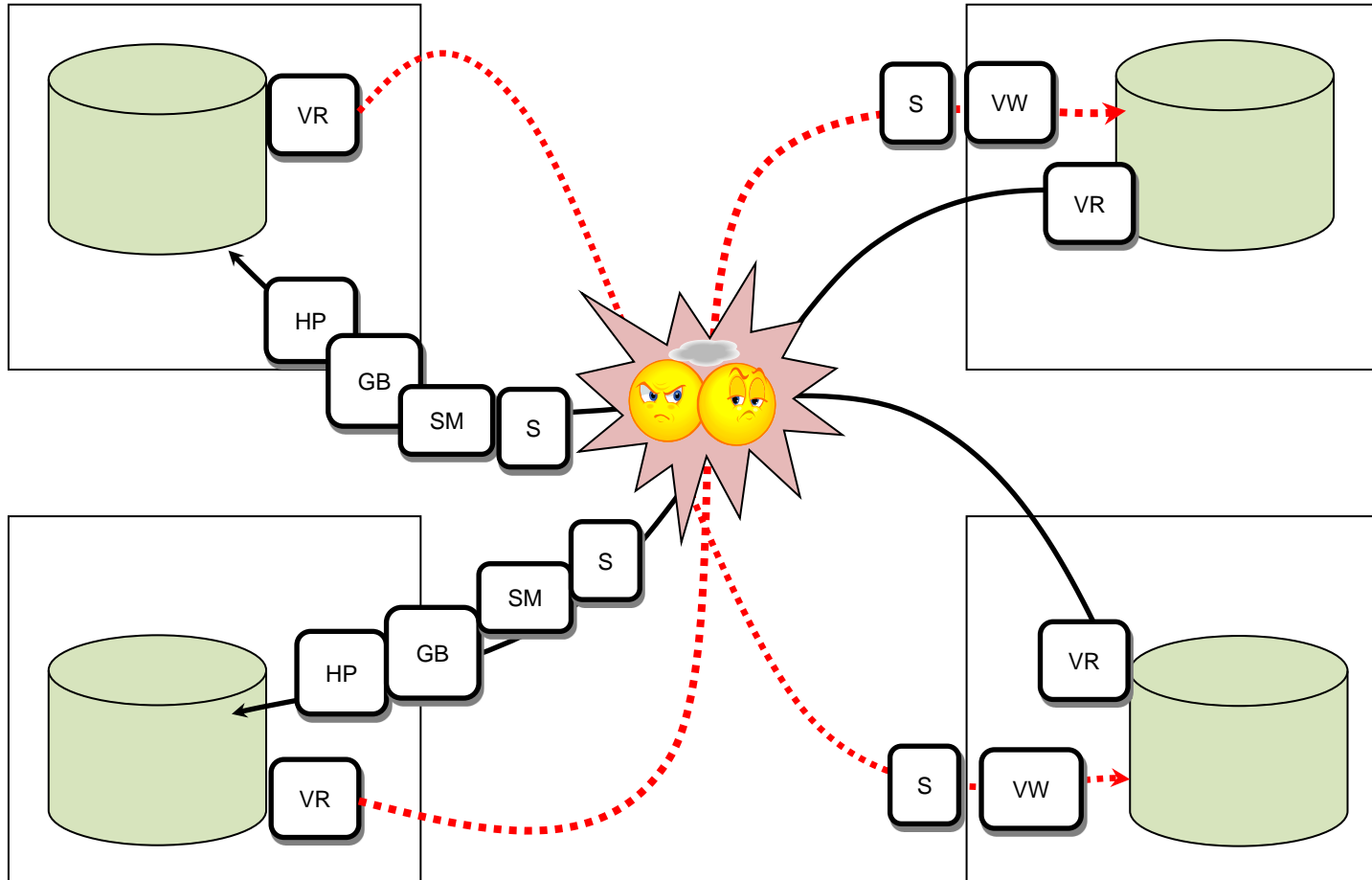


DryadLINQ – Streamlined processing





Executing Data Parallel Jobs





Challenges

- Predictable end-to-end performance
 - Fair-share
 - Differentiated priority
 - Deadline-based execution
- Good cluster utilization

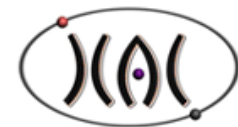




How to schedule Dryad jobs?

- **Exclusive, batch execution**
 - FCFS queue (e.g., Hadoop on demand, DryadLINQ)
 - Exclusive allocation per user/jobs
 - Cons 1: Unpredictable wait time
 - Cons 2: Low cluster utilization





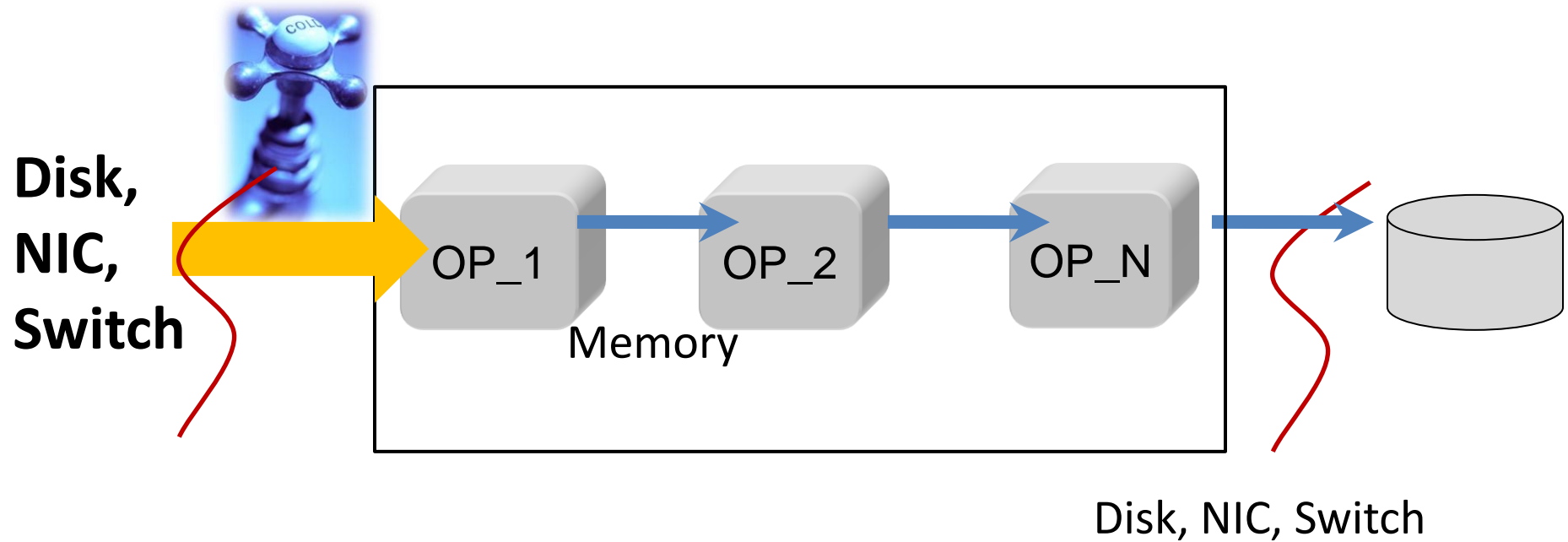
How to schedule Dryad jobs?

- **Concurrent execution w/ scheduler policy**
 - Enterprise policy (fair-share, diff. priority)
 - Facebook's Hadoop fair scheduler, Quincy from MSR
 - Cons 1: Should kill a long-running task
 - Cons 2: Don't care about how much bandwidth a task actually consume





Our approach: **throttle**, after schedule

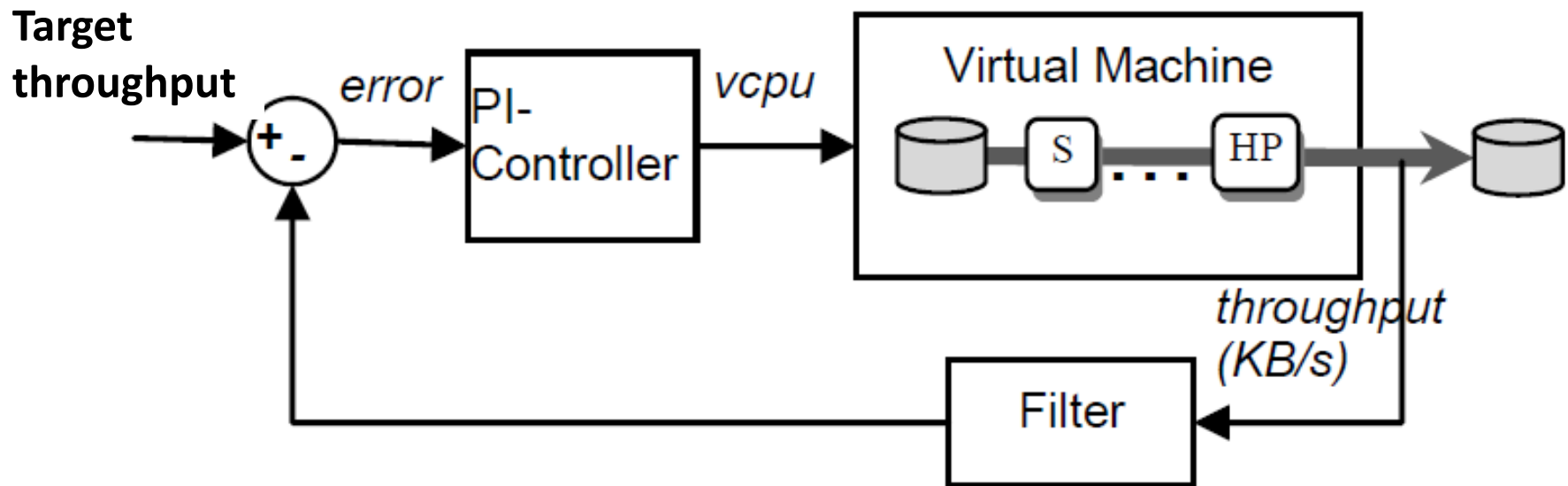


Throttle-up for my job
Throttle-down for other's jobs






Feedback control of throughput





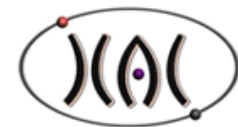
Sensing a job's progress

- Insert “sensors” during compilation step

<pre>Func__A() { Select(..) .SelectMany(..) .GroupBy(..) .HashPartition(..); }</pre>		<pre>Func__A'() { Sense(..) .Select(..) .SelectMany(..) .GroupBy(..) .HashPartition(..); }</pre>
--	--	---

- Fine-grained sensing (e.g., lines, words, objects)
- Notify throughput to controllers

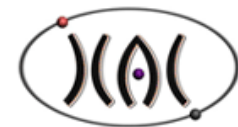




Actuation via VCPU

- Use Hyper-V's VCPU interface
- Set fraction of CPU time a VM can consume (e.g., 50% of 4 cores)
- Good linear fit between VCPU setting and DryadLINQ throughput
 - How?
- Alternative approach for controlling I/O performance





System Identification

- Profiling DryadLINQ operators running in VMs
- First-order model (T: throughput, C: VCPU)

$$T(k) = aT(k - 1) + bC(k - 1)$$

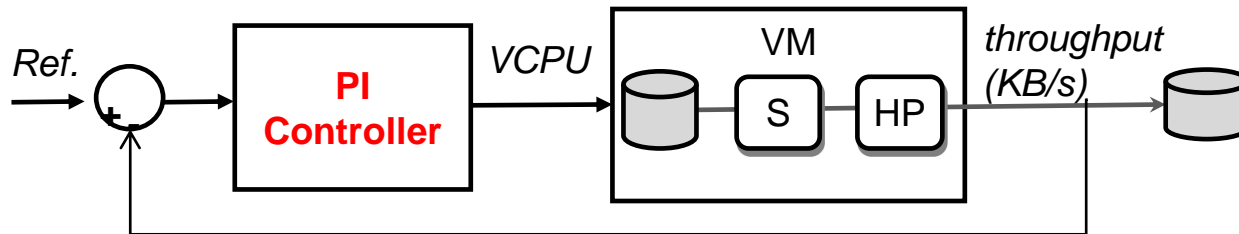
- Least-square regression to derive a model

	Parameters	R2	Steady-state at 100% VCPU
Select	a=0.26, b=102.73	0.85	52.2 MB/s
GroupBy	a=0.05, b=47.31	0.74	2.5 MB/s
OrderBy	a=0.13, b=57.13	0.87	3.3 MB/s
Join	a=0.32, b=187.54	0.93	13.8 MB/s





Proportional, Integral Controller

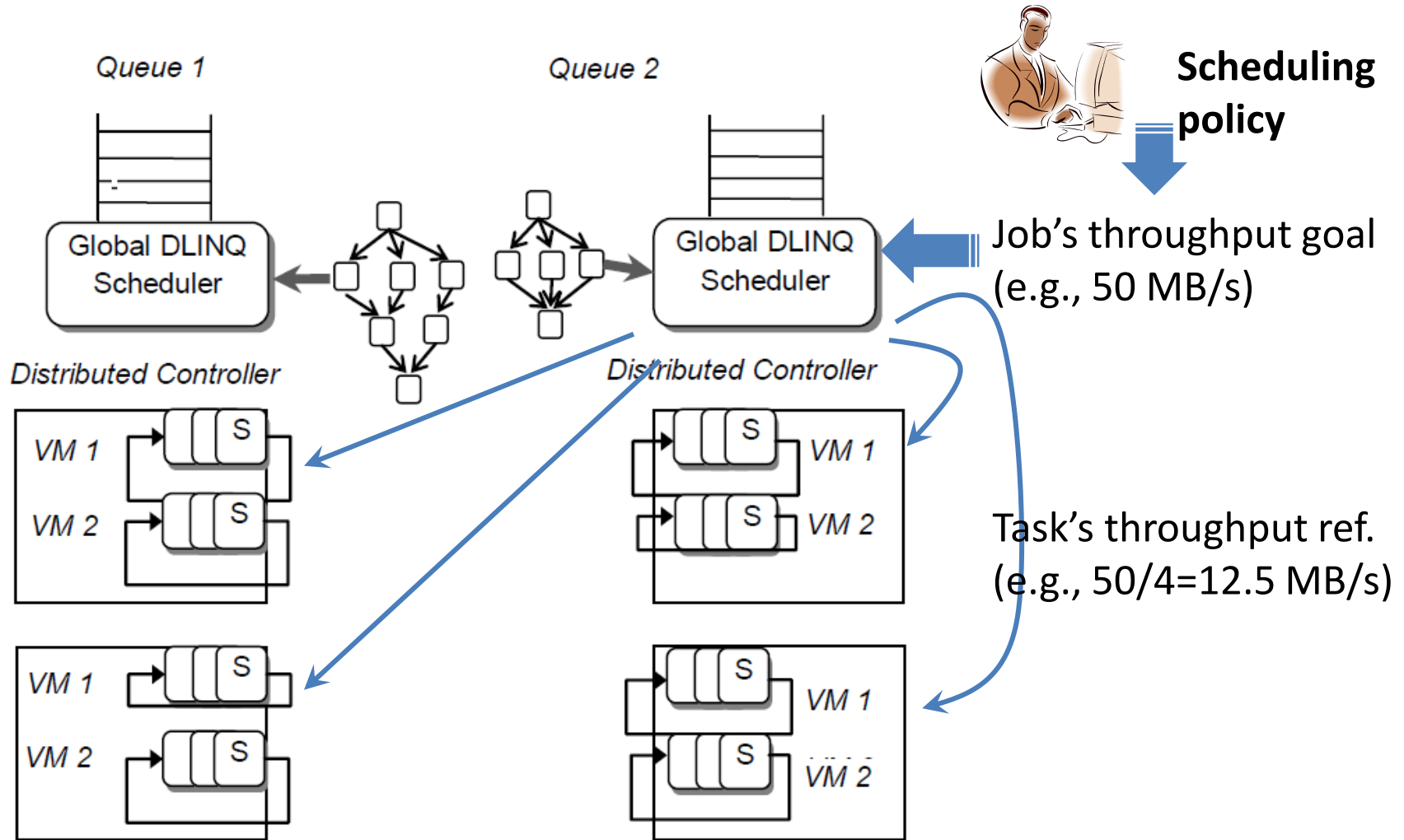


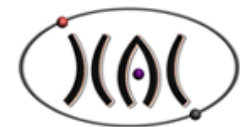
- PI controller properties
 - Proportional term: fast control
 - Integral term: 100% accuracy
- Stability
 - Should be careful for data-intensive workloads
- Controller speed
 - Theory and practice match





Distributed Controllers

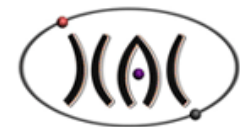




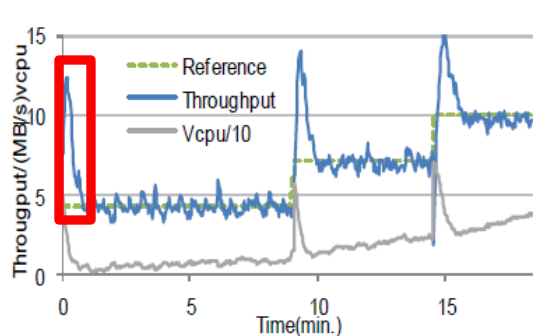
Evaluation

- 5 DryadLINQ applications
 - *Grep, GetTopK, Join, TeraSort*
 - *SkyServer*: real E-Science application
- Home-brew cluster
 - 4 compute nodes, 8 cores each
- Microsoft HPC cluster with Hyper-V
- Modified DryadLINQ academic version

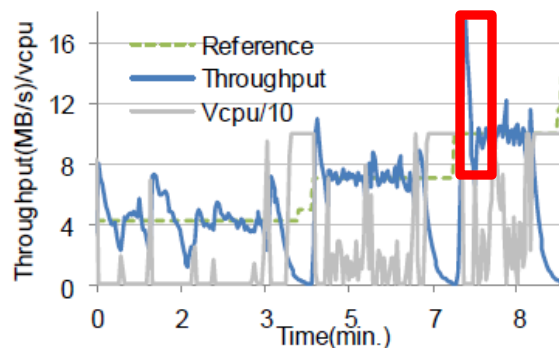




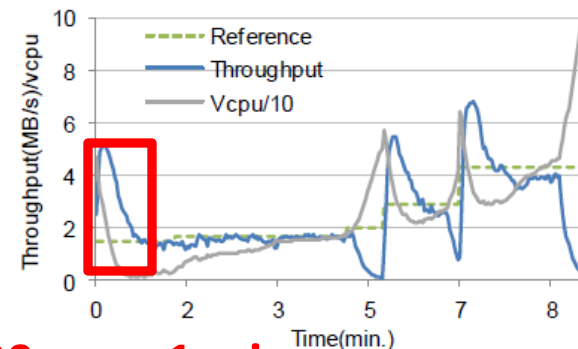
Results – Task Throughput



(a) Grep: *Select, Where*

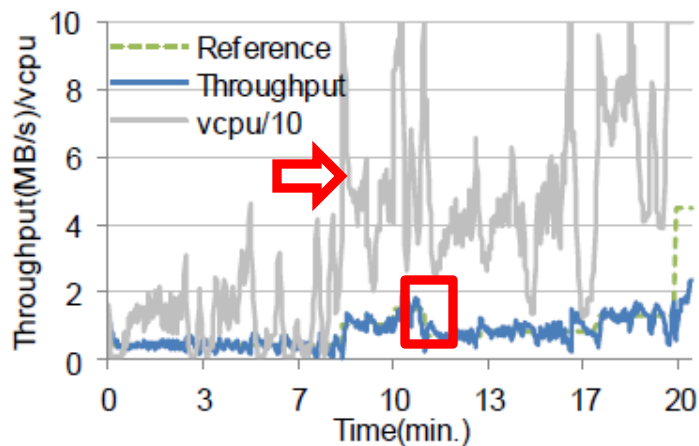


(b) TeraSort: *Merge, OrderBy*

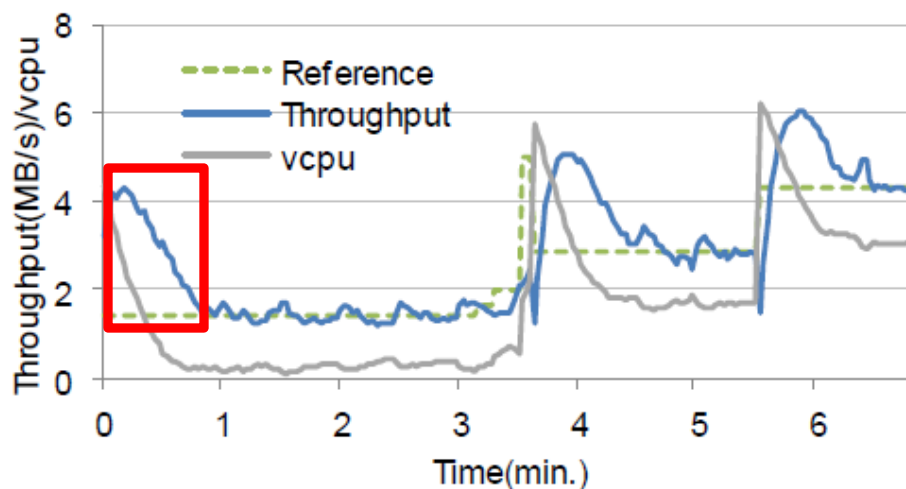


(c) GrepByJoin: *Join*

30 sec – 1 min.



(d) GetTopKWord: *Select, SelectMany, GroupBy, HashPartition*

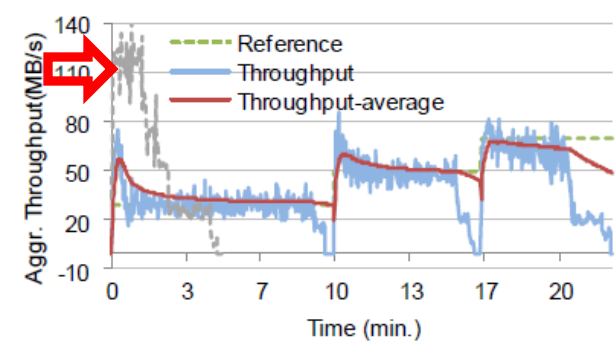


(e) SkyServer: *Join, Where, Select, Distinct, HashPartition*

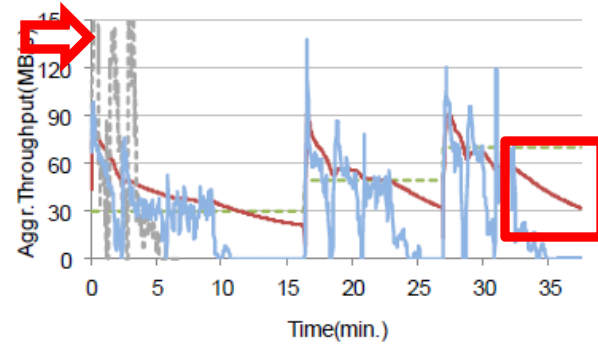




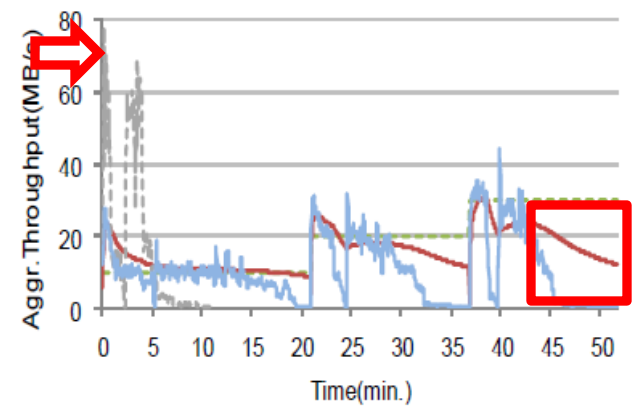
Results – Job(aggregate) throughput



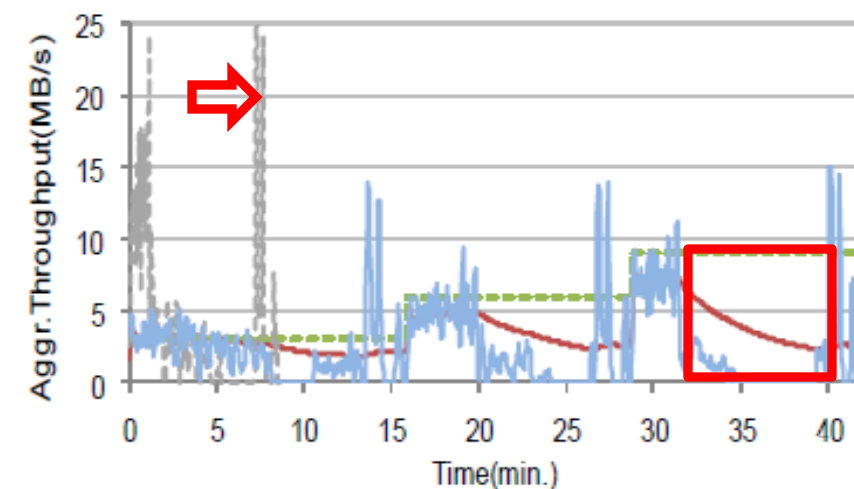
(a) Grep



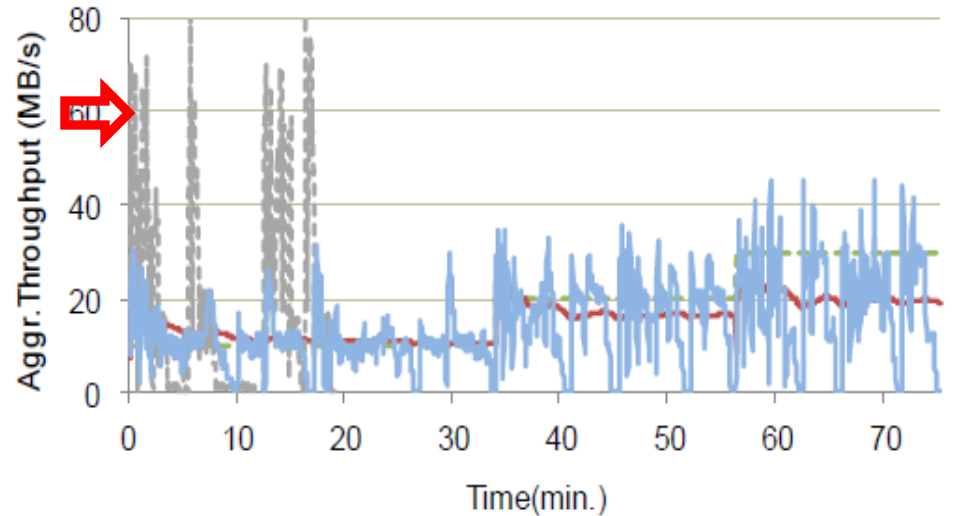
(b) TeraSort



(c) GrepBvJoin



(d) GetTopKWord



(e) SkyServer





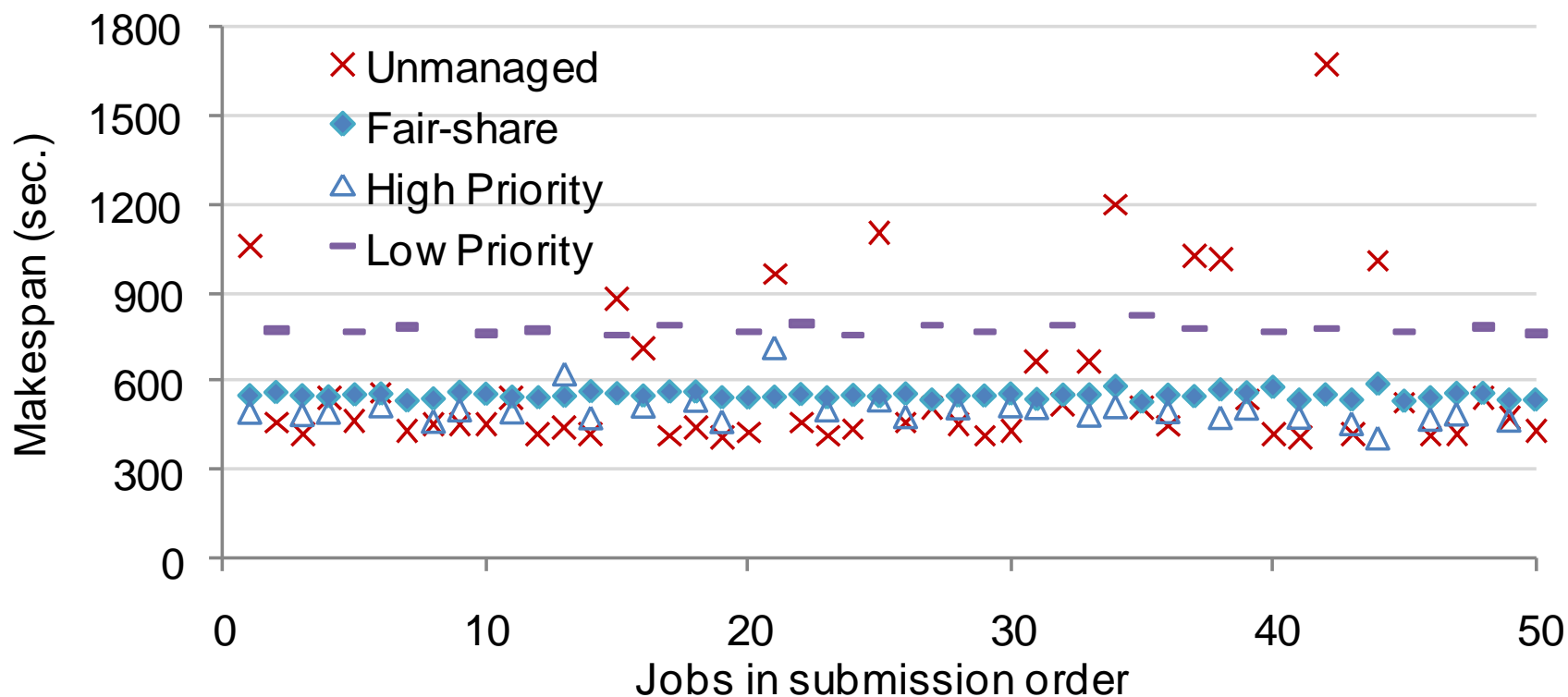
Evaluation– Scheduling Policy

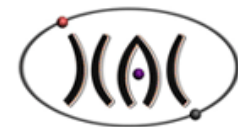
- 2 Scheduling Policies
 - **Fair-share policy**: jobs get equal share of cluster's capacity (100MB/s in the prototype)
 - **Diff. priority policy**: jobs get 2/3 & 1/3 of capacity
- SkyServer as the use-case
 - 50 job instances, each with 72 tasks
 - 20 hours running time





Results – Policy Implementation





Comparisons

	Scheduler approach		
	Exclusive, batch scheduler	Fine-grained, multi-job scheduler	Throttling (ours)
Scheduling granularity	Per-Job	Per-task	Per-byte throughput
Scheduler policy	Hard	flexible	Very flexible
Difficulty to implement	Simple	Complex	Complex (?)

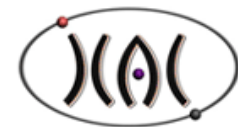




Conclusion

- Control-theoretic approach to MapReduce-style computation
- VCPU as actuation knob for data throughput
- PI controller shown to work well
- Complementary to existing cluster schedulers





Discussion

- Usability
 - System ID and controller design for only few standard LINQ operators
 - Need further study with more apps
- Distributed Controllers
 - How to predict and eliminate undesirable interactions?





Thank you!

