

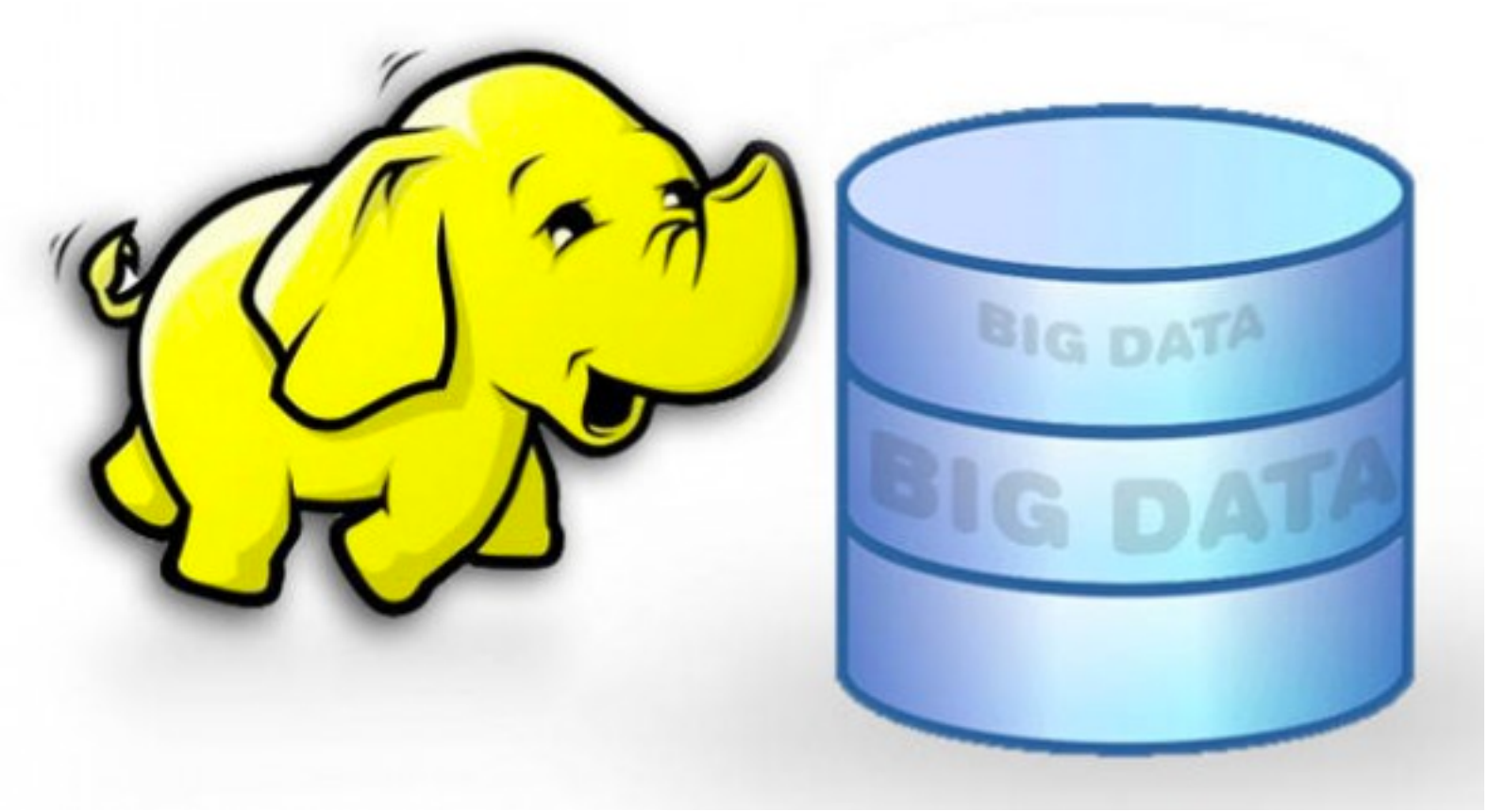


Bistro: Scheduling Data-Parallel Batch Jobs against Live Production Systems

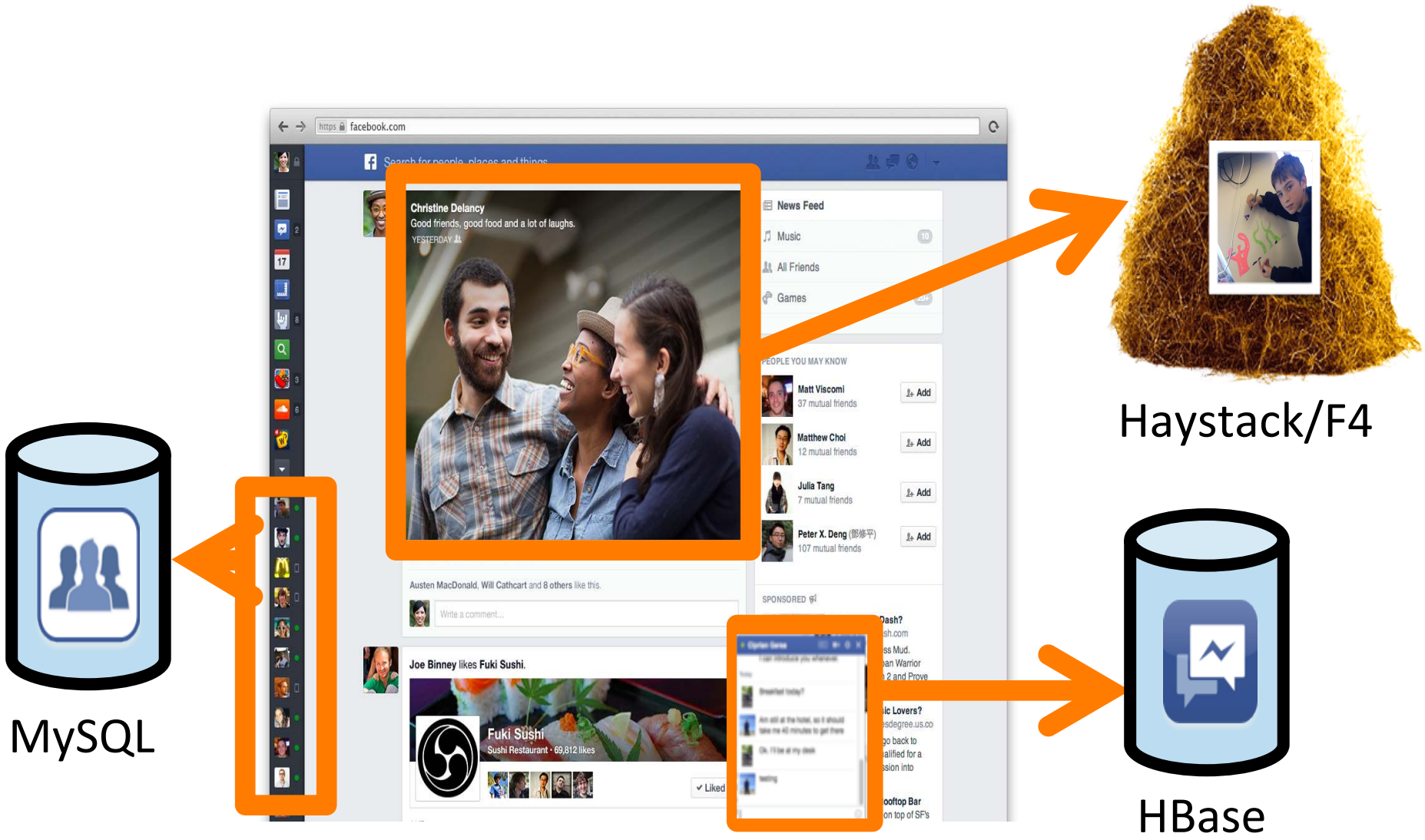
http://bistro.io

Andrey Goder, Alexey Spiridonov, Yin Wang
(Facebook)

Big Data and Hadoop



Facebook Data Store



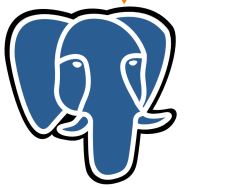
Facebook Data Store (cont'd)





RocksDB





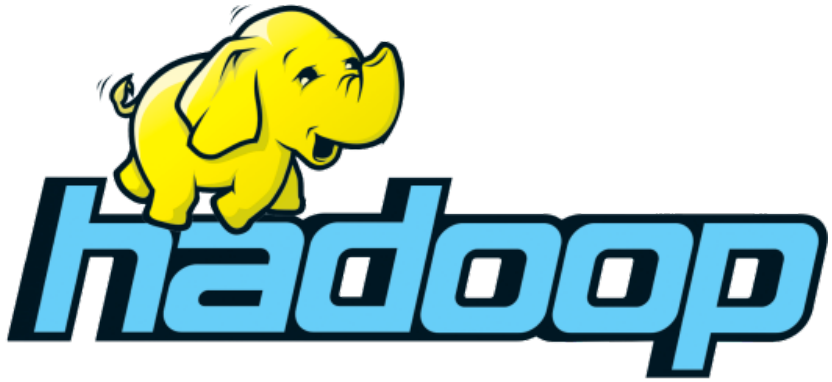
PostgreSQL

Online vs. Offline









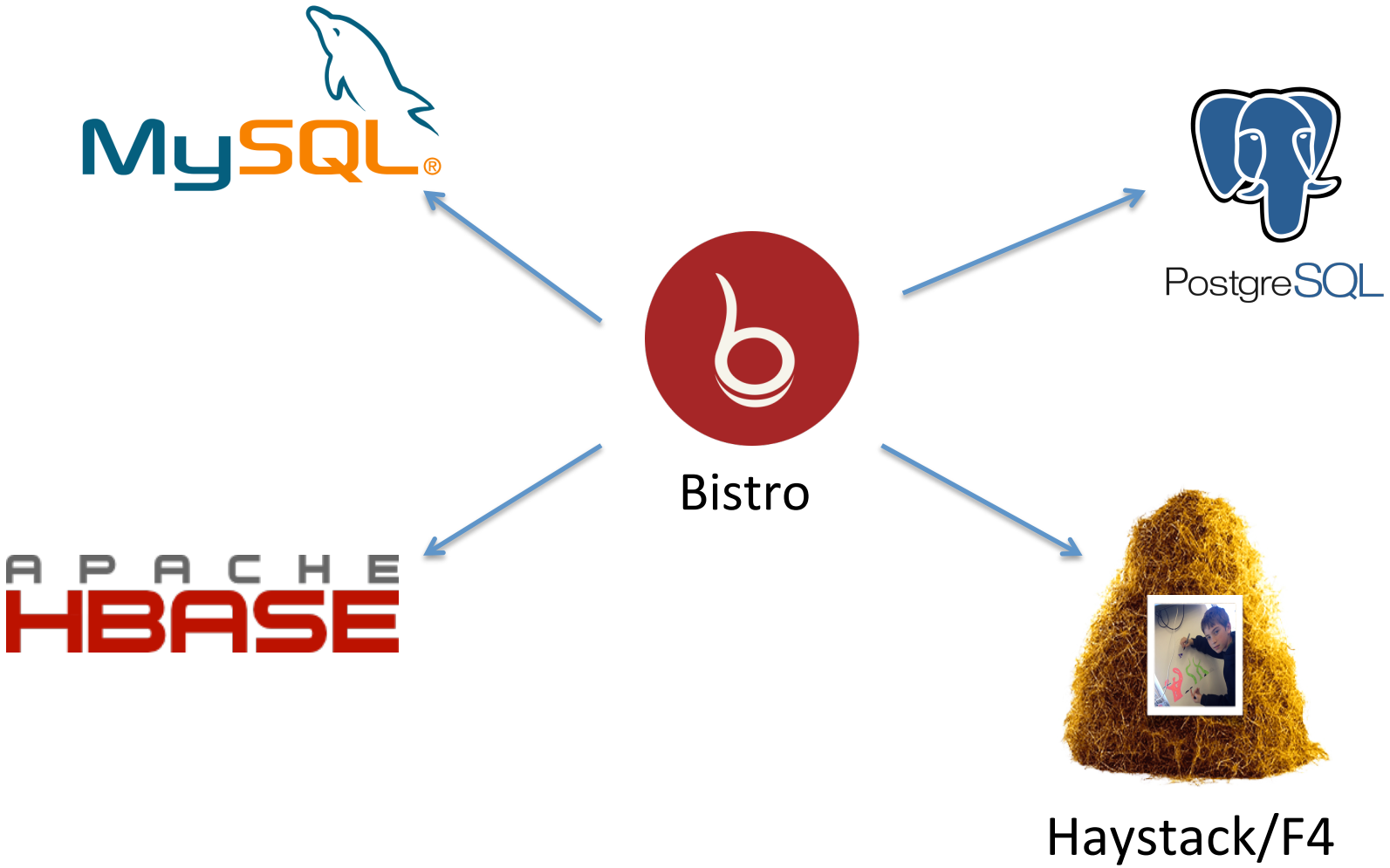


Haystack/F4



RocksDB

Bistro: Scheduling against a Variety of Online Systems







Outline

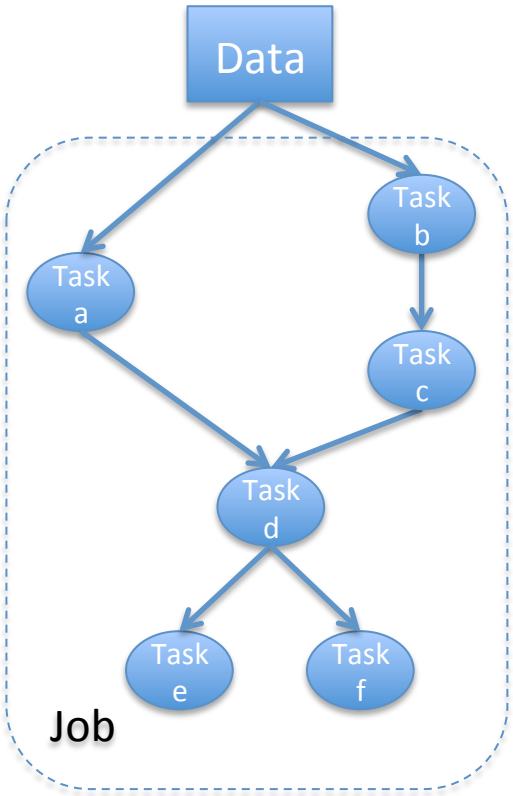
- Background
- The scheduling problem
- Bistro implementation
- Case studies

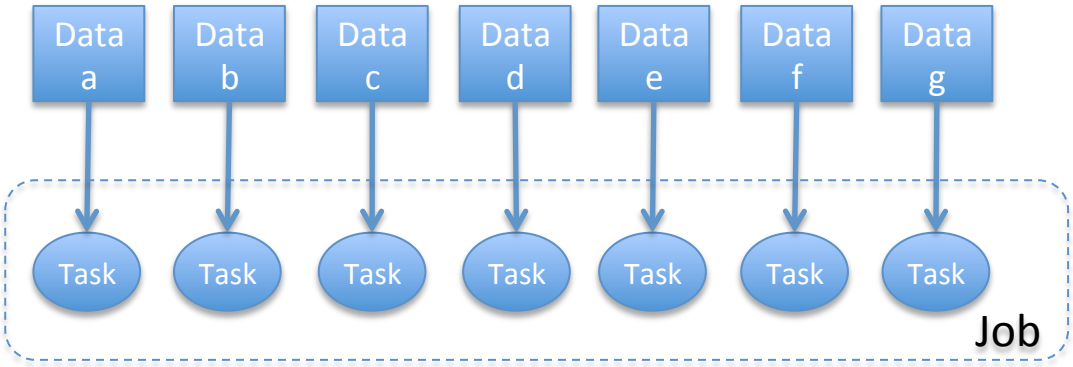




Data-Parallel Jobs

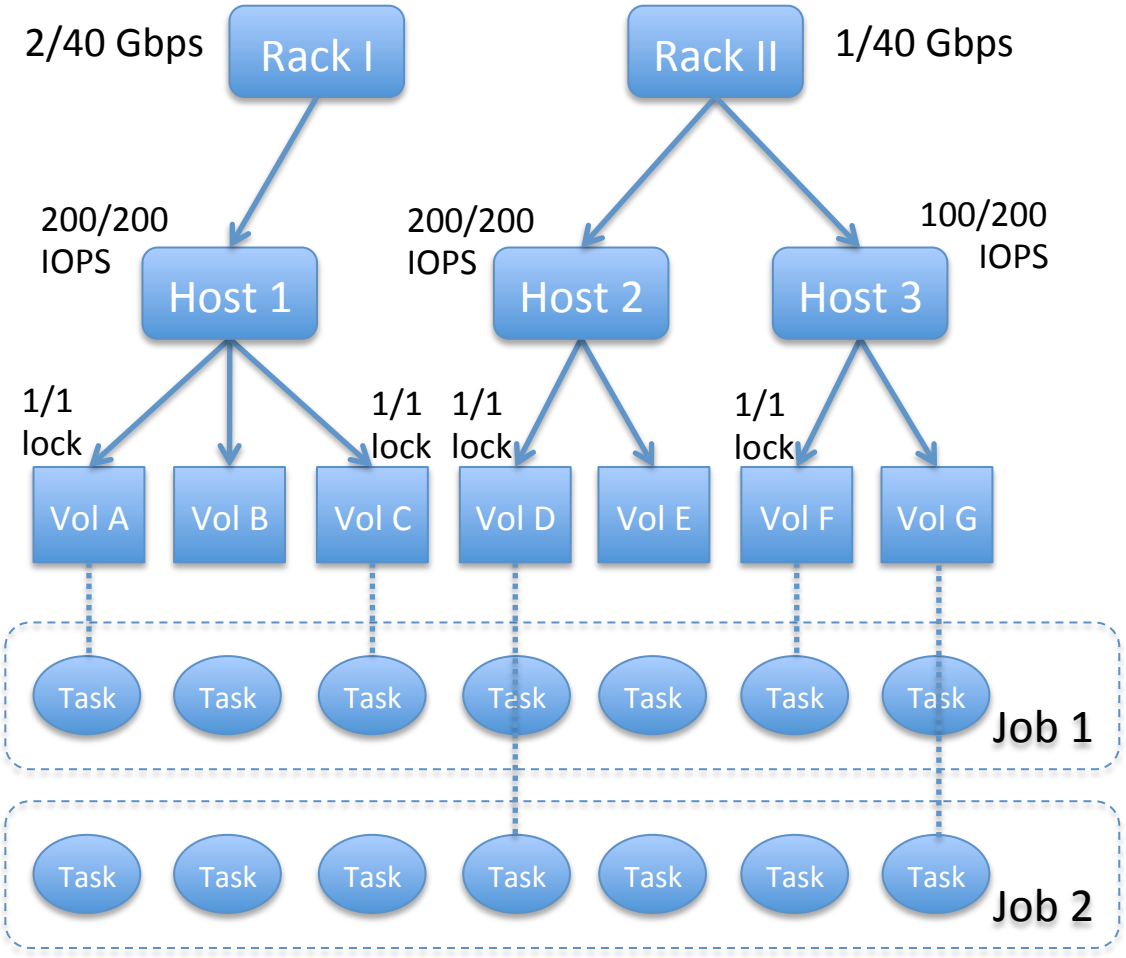
- Task-Parallel vs. Data-Parallel





Map-only jobs

The Scheduling Problem



Goal: maximize resource utilization (high throughput)

Challenge: minimize scheduling overhead at scale

./photo_process
1 lock, 100 IOPS, 1 Gbps

./volume_compact
1 lock, 200 IOPS, 0 Gbps





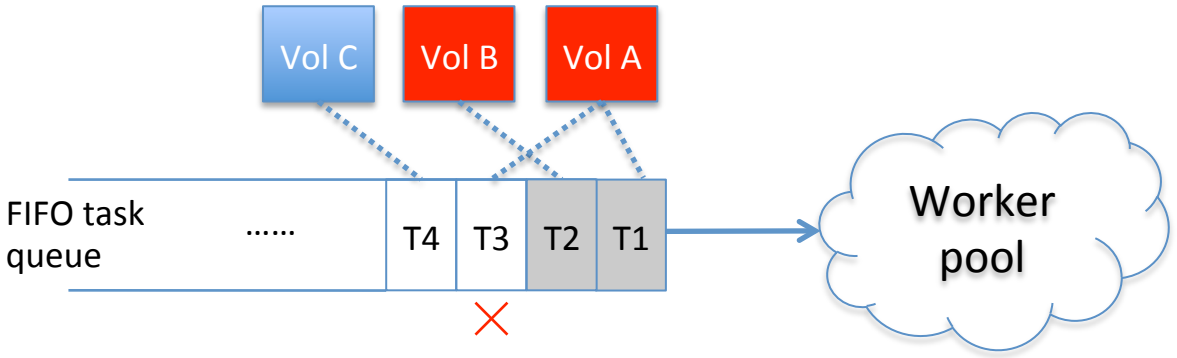
Existing Schedulers

- Online schedulers
  - E.g., load balancing, low latency, quality of service
  - Not designed for data-parallel batch jobs
- Offline schedulers
  - Can easily overload a data host
    - Focus on computation resources on worker hosts, ignoring resource constraints on data hosts
  - Little support for mutable data
  - Tightly integrated with a specific offline system

Fundamental problem: queue-based scheduling!

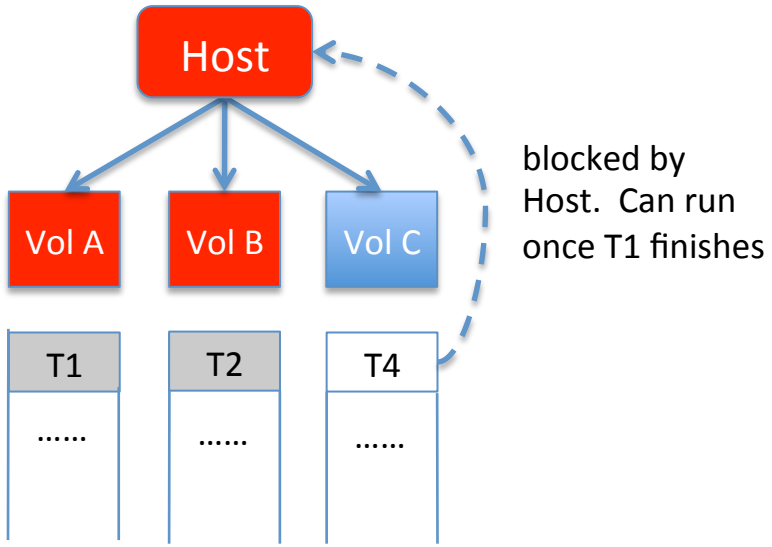
Head-of-Queue Blocking

Global queue?



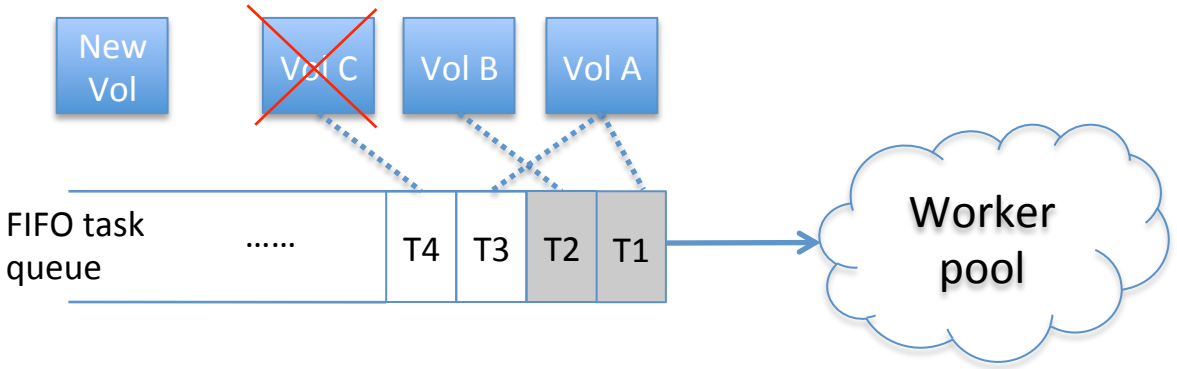
T3 blocks the rest of the queue!

One queue per shard?



blocked by Host. Can run once T1 finishes

How to Track Changes







Scheduling Algorithm – Brute-force

```
while true:
    for j in jobs:
        for n in leaf nodes:
            if t has not finished and there are enough
              resources for j along the path to root:
             run(t)
```





Scheduling Algorithm – Subtree

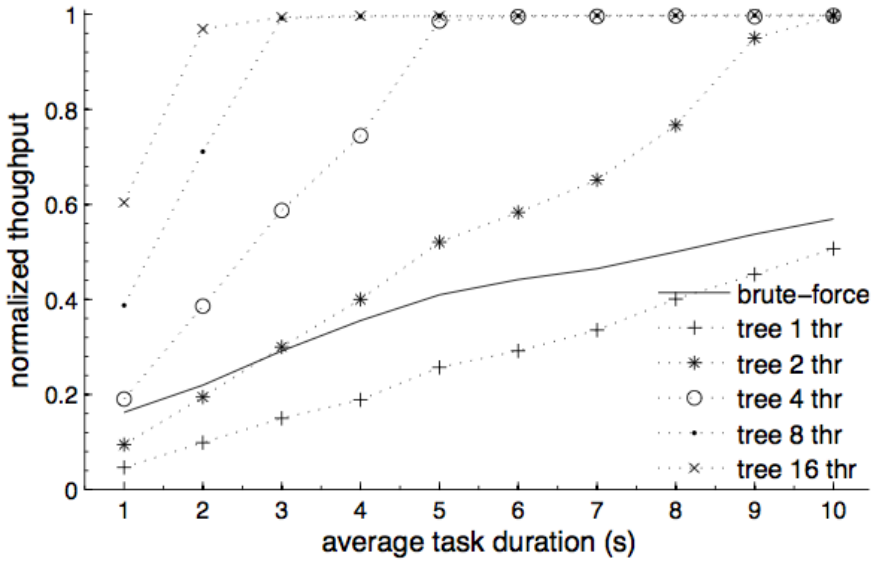
when a task *t* finishes at leaf node *n*, run:

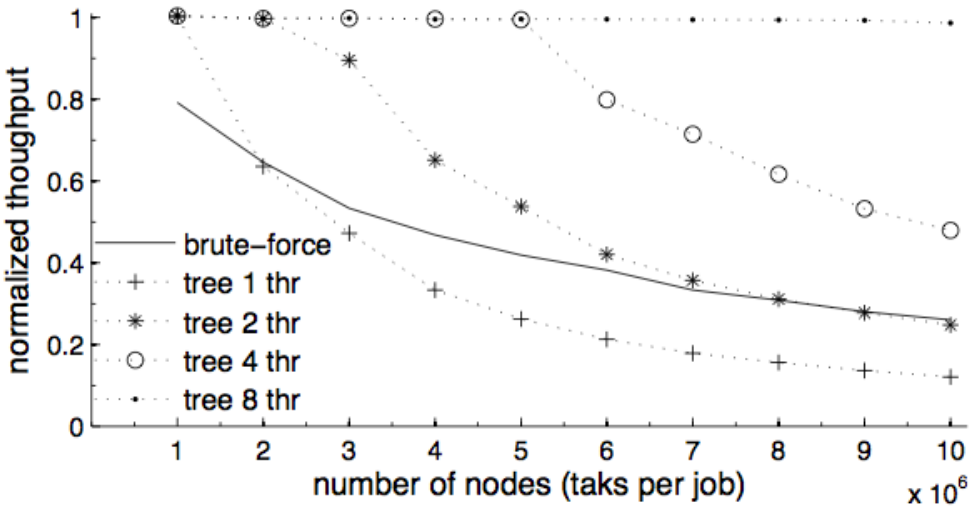
p = the highest ancestor node of *n* where *t* consumes resources

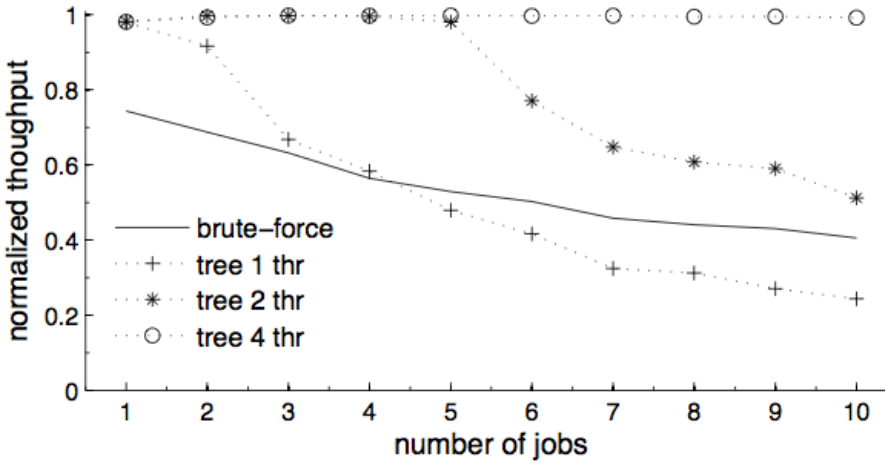
run brute-force algorithm on the subtree at *p*

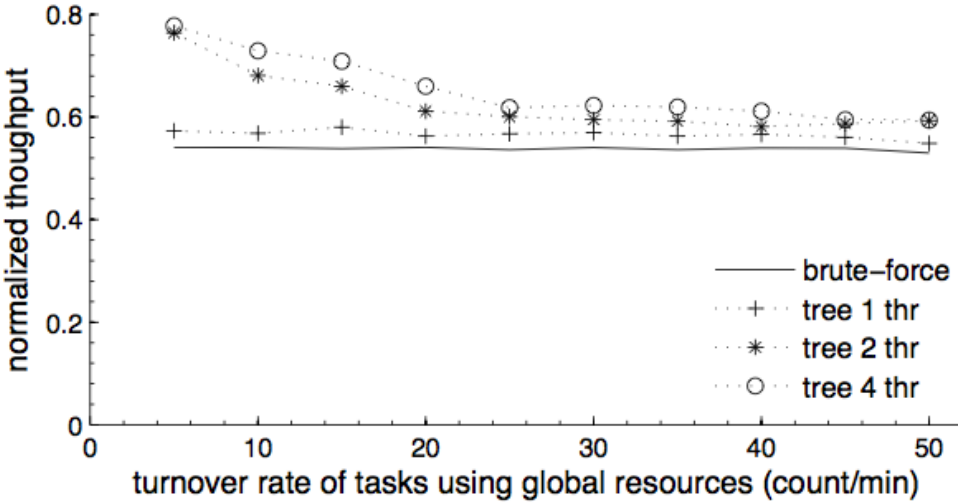
Enables parallel scheduling!

Micro-benchmark









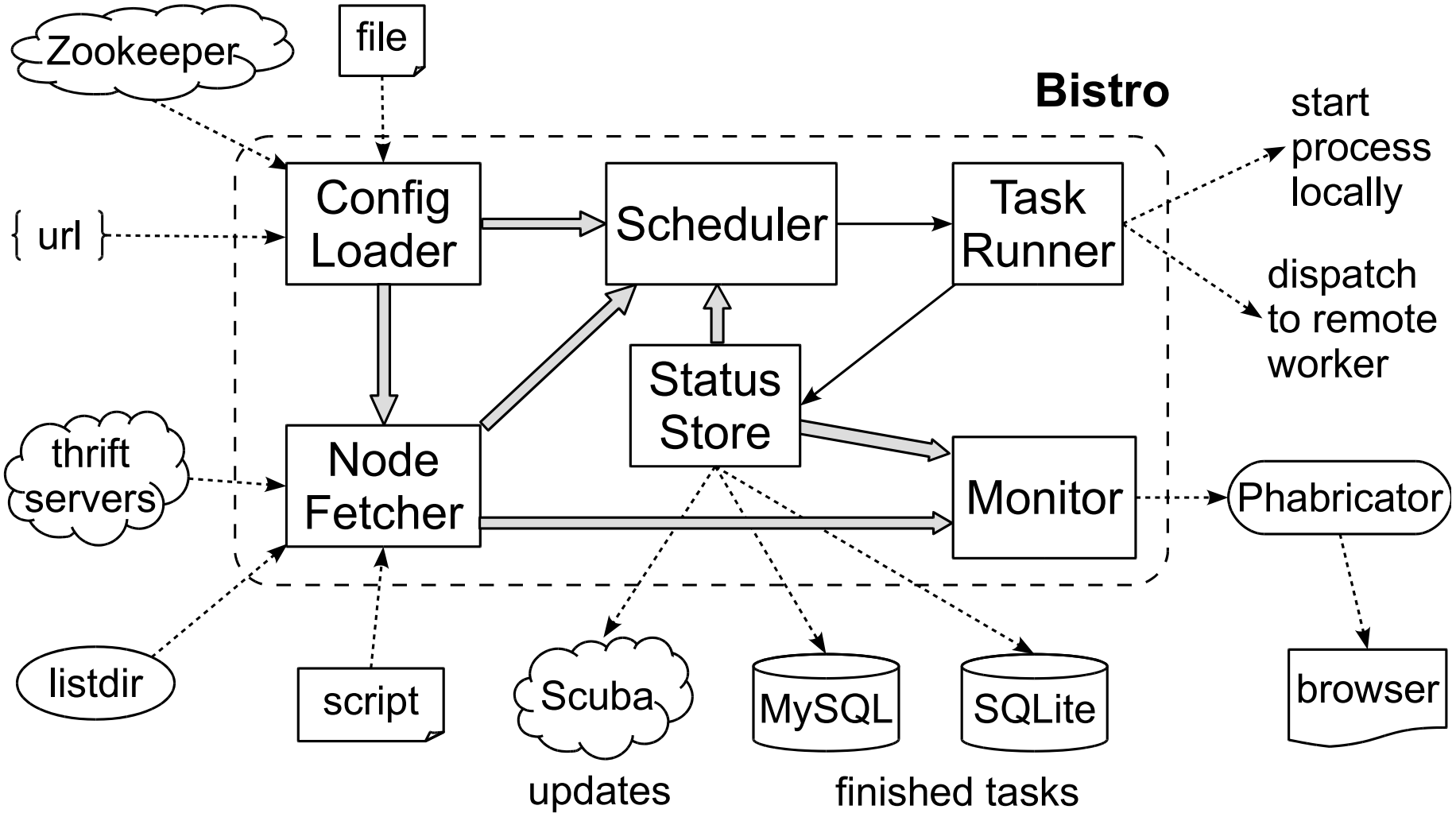




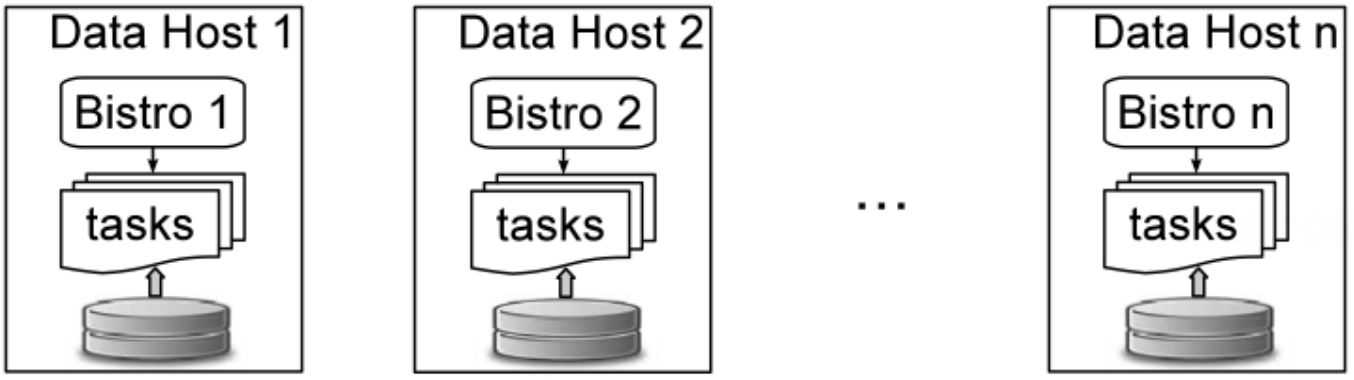
Outline

- Background
- The scheduling problem
- Bistro implementation
- Case studies

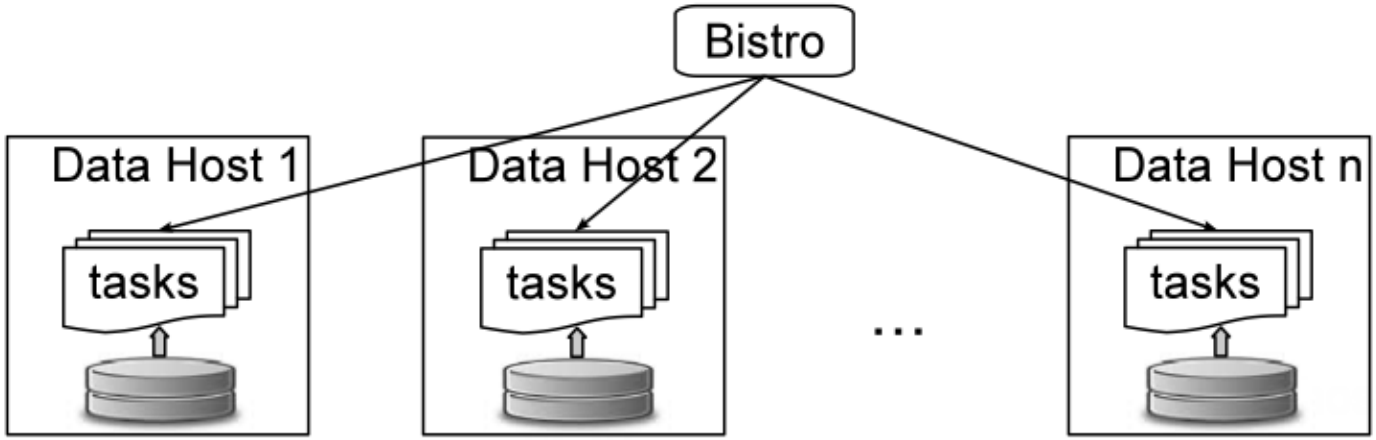
Architecture



Scheduling Modes

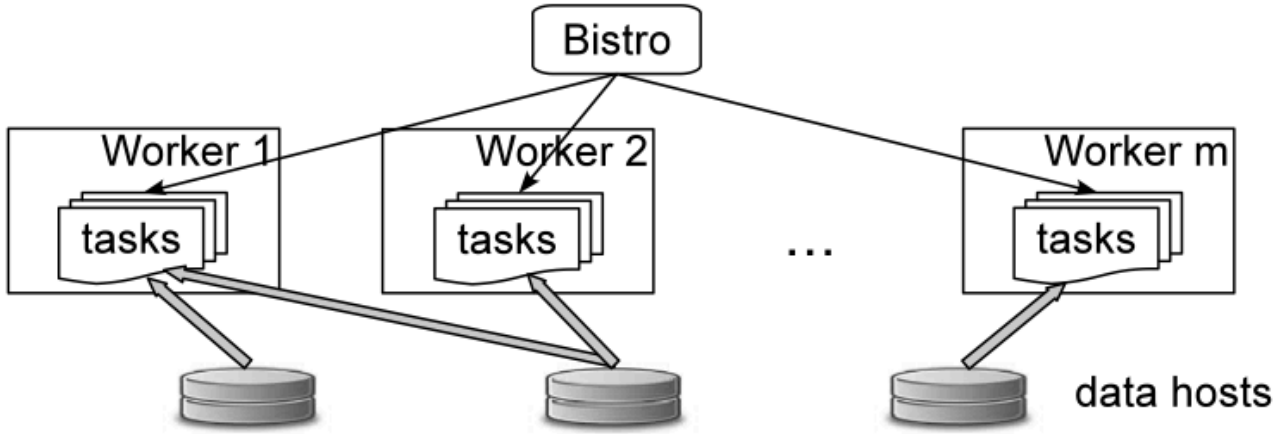


(a) **multi/co-locate**

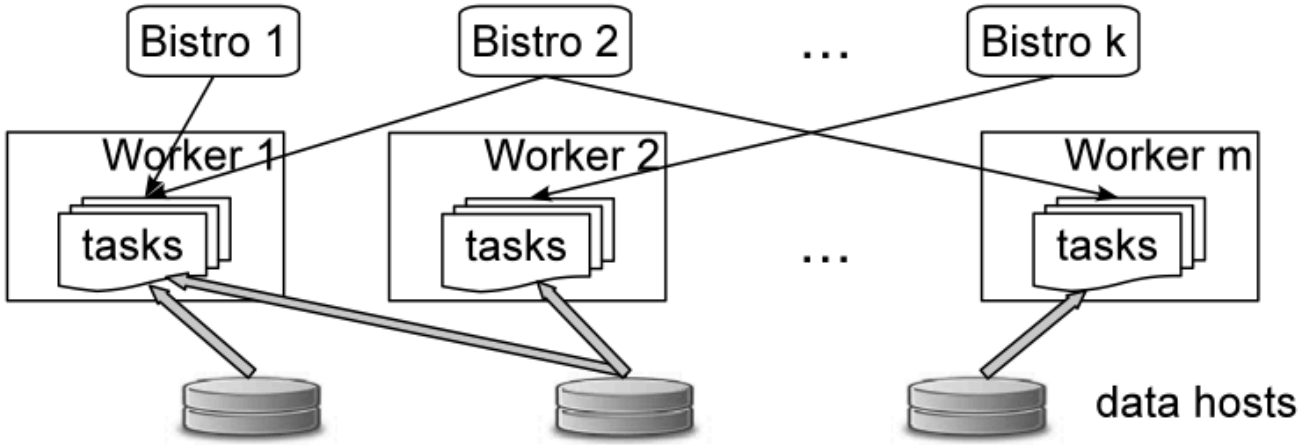


(b) **single/co-locate**

Scheduling Modes (cont'd)



(d) **single/remote**



(e) **multi/remote**





Configuration

```
"nodes": {
  "levels": ["rack", "host", "volume"],
  "node_fetchers": [
    {"source": "haystack_racks"},
    {"source": "haystack_hosts"},
    {"source": "haystack_volumes",
      "prefs" : {
        "volume_type" : "PHOTO"
      }
    }
  ]
},
"resources": {
  "rack": {"capacity": 20, "default": 1}
  "host": {"capacity": 200, "default": 100}
}
```

Resource config

```
"job->photo_process": {
  "cmd": "./photo_process",
  "args": {
    "model": "deep_face",
  }
  "enabled": true,
  "filters": {
    "host": {
      "blacklist_regex": "^haystack(\d+)\.prn"
    }
  },
},

"job->volume_compact": {
  …
}
```

Job config





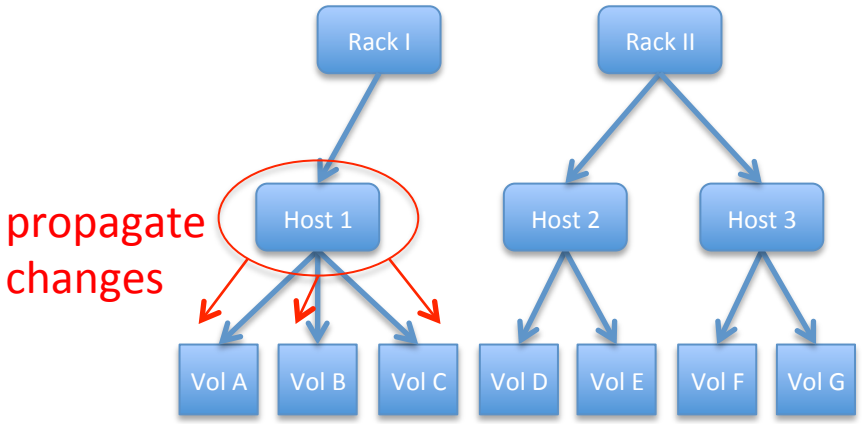
Scheduling Policies

- Round robin
- Randomized priority
- Ranked priority
- Long tail

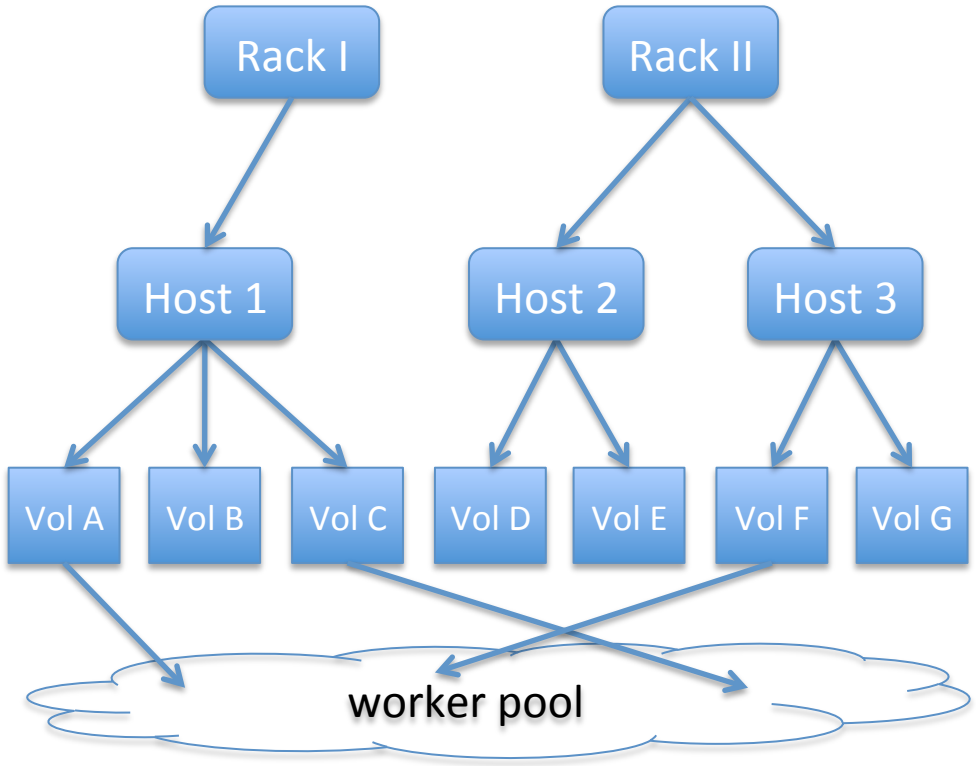
```
while true:
    for j in jobs:
        for n in leaf nodes:
            if t has enough resources
               along the path to root:
               start(t)
```

Handle Model Updates

- Resource and job updates are common
- Periodically reload configuration, and update nodes
- Propagate changes to descendants
- Asynchronous model updates, scheduling, monitoring etc.



Worker Resource Management

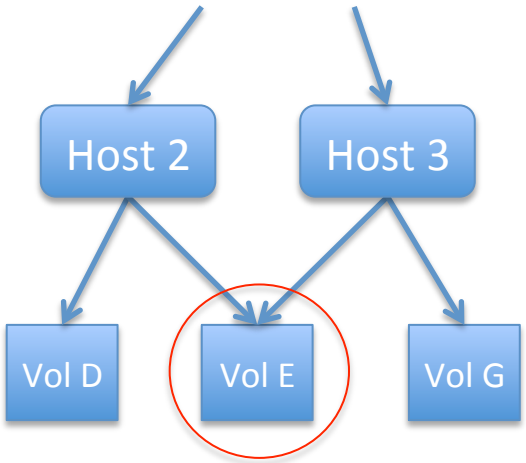



```
run(t):
  for w in workers:
    if w has enough
      resources for t:
      run(t, w)
      return
```

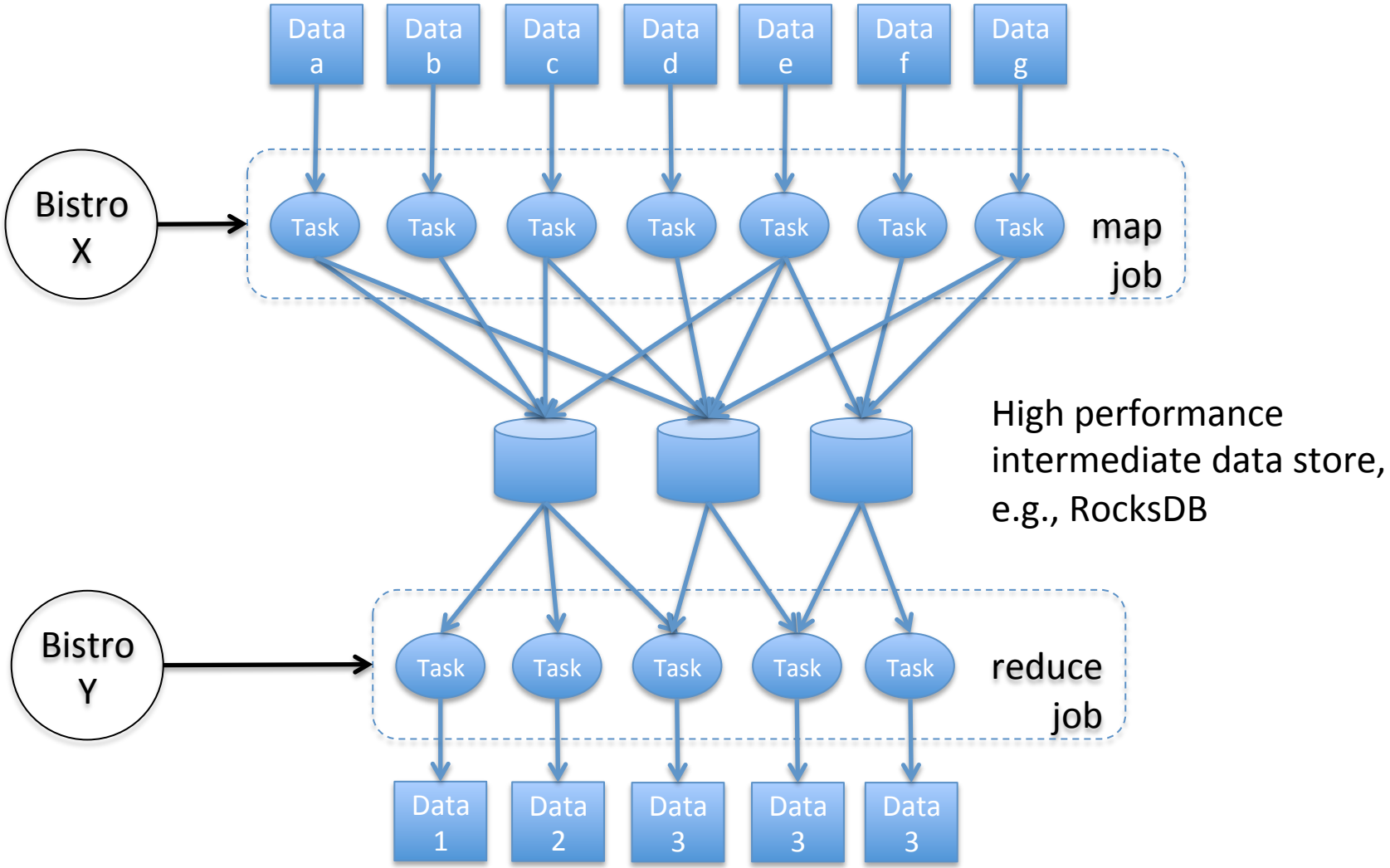
- Just another layer of nodes in resource model
- Allows both resource constraints and placement constraints
- Can use randomized algorithms for scalability

Model Extensions

- Multiple resources per node
- Execute jobs on different levels
- Time-based jobs
  - Time nodes
- Different model partitioning schemes
- DAG model
  - Model replicas



“Reduce” Support







Adaptation to Live Traffic?

- Resource monitoring + Bistro model update
- What about spikey live workload?
  - Aggressive task preemption
  - Rapid cleanup
  - Making progress under frequent preemption?
- Conservative or simple scheduling policy wins
  - Overprovisioning
  - Day/night resource capacity adjustment





Model Limitations

- More complex resource constraints
  - Source destination
    - Break into two jobs
  - Network





Outline

- Background
- The scheduling problem
- Bistro implementation
- Use cases





Use Cases

Online system	Example Job	Resource hierarchy	Replacing
MySQL	DB Iterator	root->host->db	Proprietary scheduler
PostgresSQL	Migration	host->shard	Proprietary scheduler
BLOB storage	Photo/video processing	Host->volume->file	
HBase	Data compression	host->region	Hadoop

- Our only general-purpose scheduler for non-HDFS systems
- Replacing Hadoop for many online HDFS systems

Summary

Application	Mode	Resource Model					Job			
		level	resource type	node count	concur-rency	change rate	concurrent jobs	change rate	task data	task duration
UDB Migration	single/ remote	1 2 3	root host db	1 12 k 312 k	various 2 1	- 2.1%/ hr 2.1%/ hr	5	10/ day	330k rows	5 min
UDB Scraping	single/ remote	1 2	host db	12 k 355 k	1 1	2.4%/ hr 2.3%/ hr	10	5/ hr	1 MB	5 sec
ODS Rollup	multi/ co-locate	1 2 3	host region time	100 3 k 24 k	3 3	2.6%/ min 3.1%/ min 8.3%/ min	10	10/hr	2.8 m rows	1 min
CheaFS Migration	single/ co-locate	1 2	host dir	2.3 k 163 k	3 1	<1%/ day <1%/ day	1	<1/ day	100 GB	7 hr
Photo Iterator	multi/ local	1 2	host volume	1 k 330 k	1 1	1.0%/ hr 1.0%/ hr	1	<1/ day	100 GB	10 hr
Video Re-encoding	multi/ remote	1 2 3	host volume video	1 k 146 k 2.12 b	1 1 1	1.3%/ hr 1.2%/ hr 1.3%/ hr	1	<1/ day	6.2MB	20 min

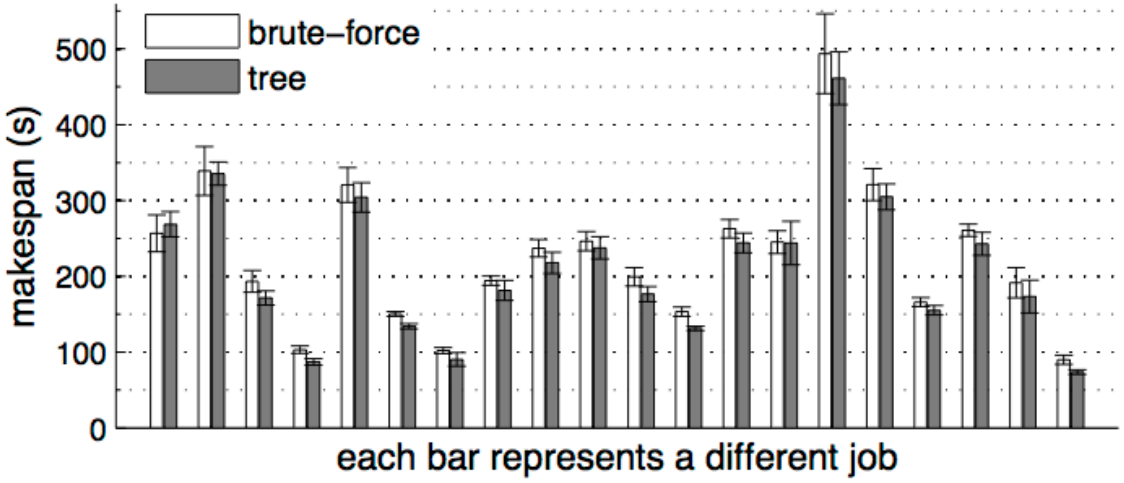




SQL Database

- Database Iterator
  - Motivating use case
  - Largest user base
- Database scraping
  - Shortest duration

Brute-force vs. Tree scheduling for scraping workload



(a) Makespan of scheduling one job



(b) Makespan of scheduling multiple jobs concurrently

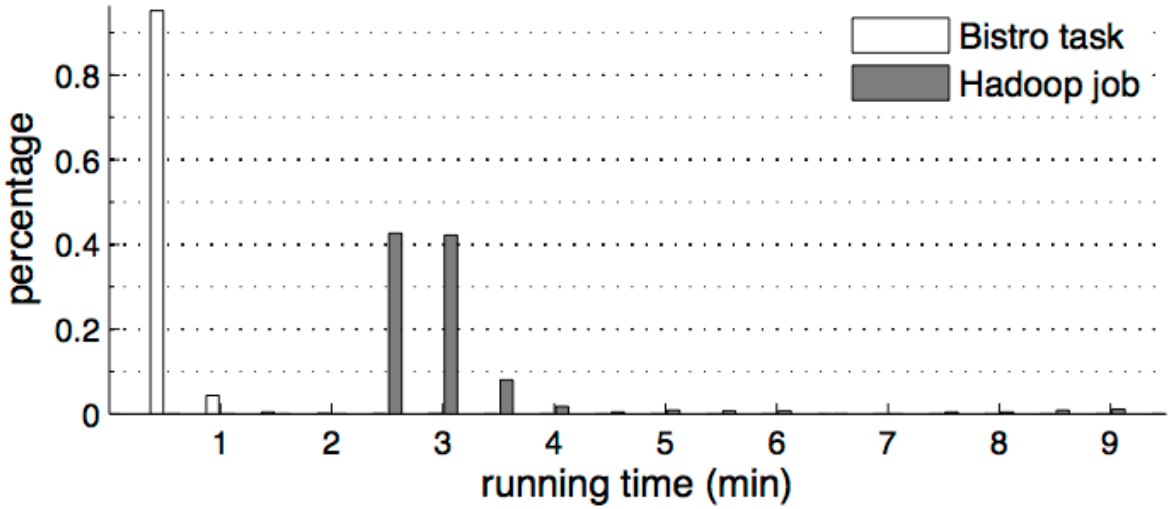




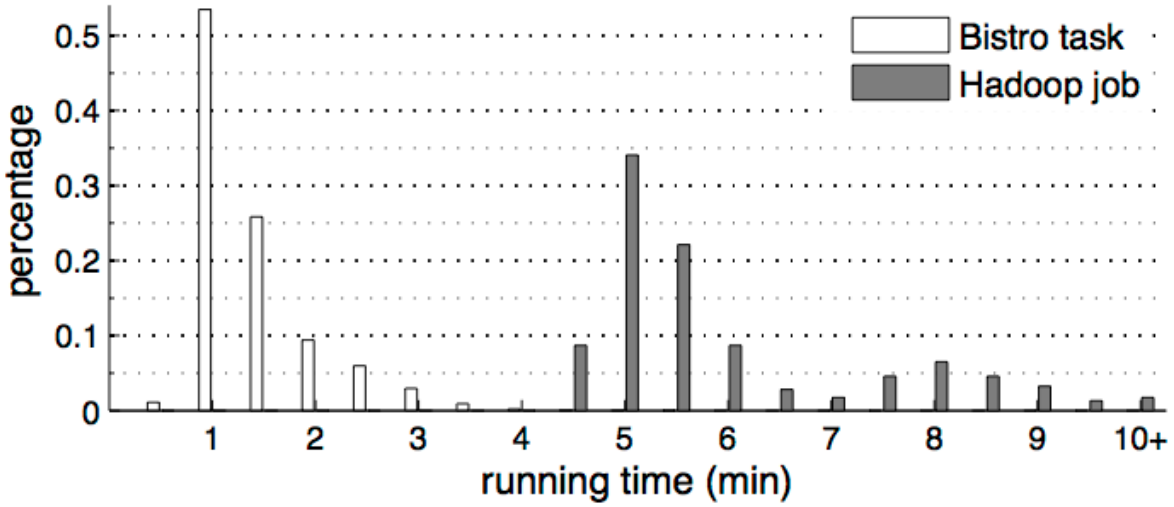
Hbase: Problems with Hadoop

- Designed for offline batch jobs, no protection for live traffic
- Very little control over job execution
  - No canary, pause/resume, debug/re-execute checkpoint etc.
- Rigid computation model
  - All or nothing
  - Barrier of each phase

Hbase Compression (cont'd)



(a) HBase table A



(b) HBase table B





Conclusions

- Running jobs directly against production systems is the trend
- Tree-based scheduling allows
  - Hierarchical resource constraints
  - Efficient updates
  - Easy partitioning for flexible setup
  - Parallel scheduling
- Open source at http://Bistro.io