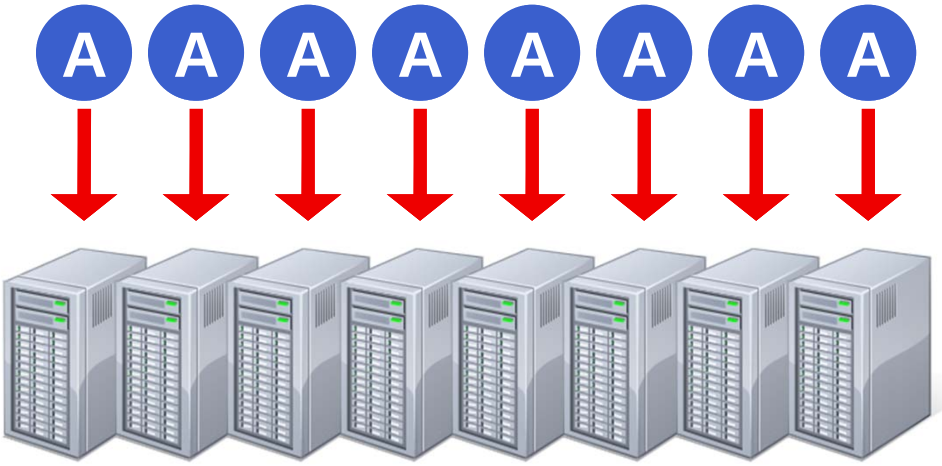
CIEL
Control-flow in a distributed execution engine

Derek Murray
University of Cambridge
10th February 2011

Distributed execution is ~~hard~~ tricky

1. Data and code distribution
2. Communication
3. Fault tolerance
4. Scheduling

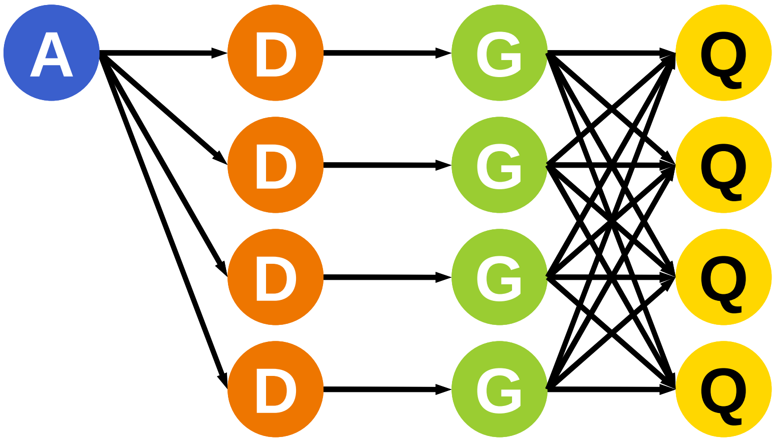
Task parallelism



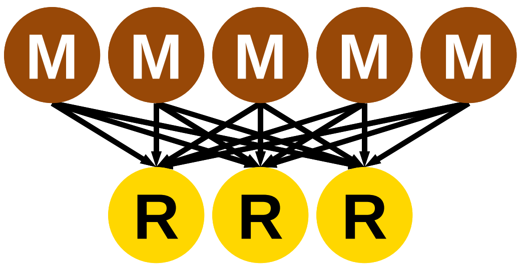
Distributed execution engines

1. Data and code distribution
2. Communication
3. Fault tolerance
4. Scheduling

Task dependencies/data-flow

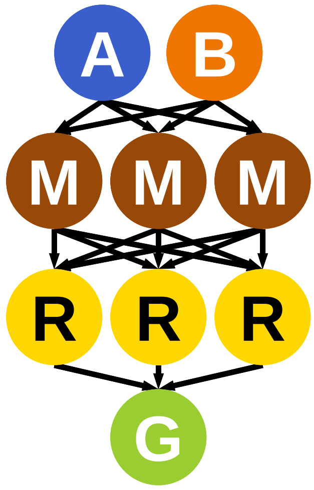


MapReduce



Dryad

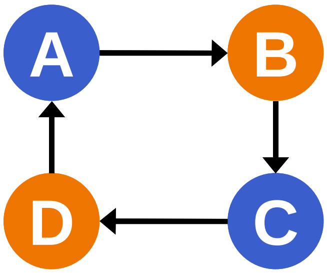
Jobs represented as:
- directed **acyclic** graphs
- **static** graphs



Distributed execution engines

1. Data and code distribution
2. Communication
3. Fault tolerance
4. Scheduling
5. Dependency tracking

Cyclic dependencies?




```
while (!converged) {
    // do stuff
}
```

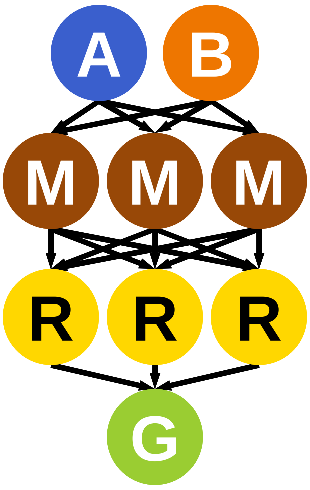
Distributed execution engines

1. Data and code distribution
2. Communication
3. Fault tolerance
4. Scheduling
5. Dependency tracking
6. **Control flow?**

Dryad

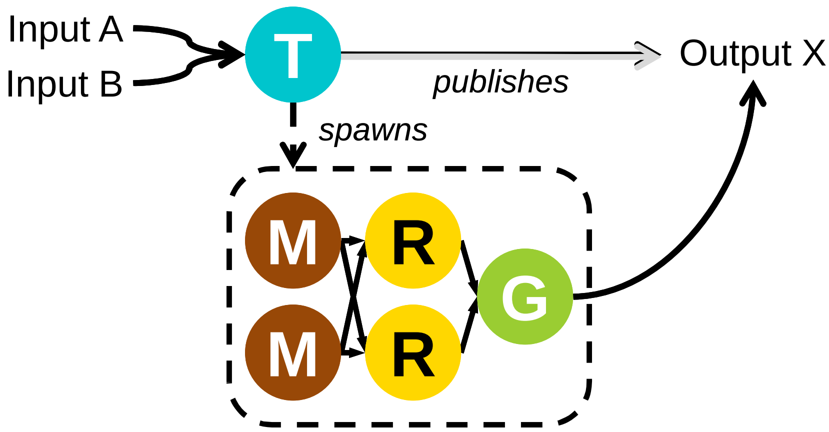
Jobs represented as:

- directed **acyclic** graphs
- **static** graphs



Idea: dynamic task graphs

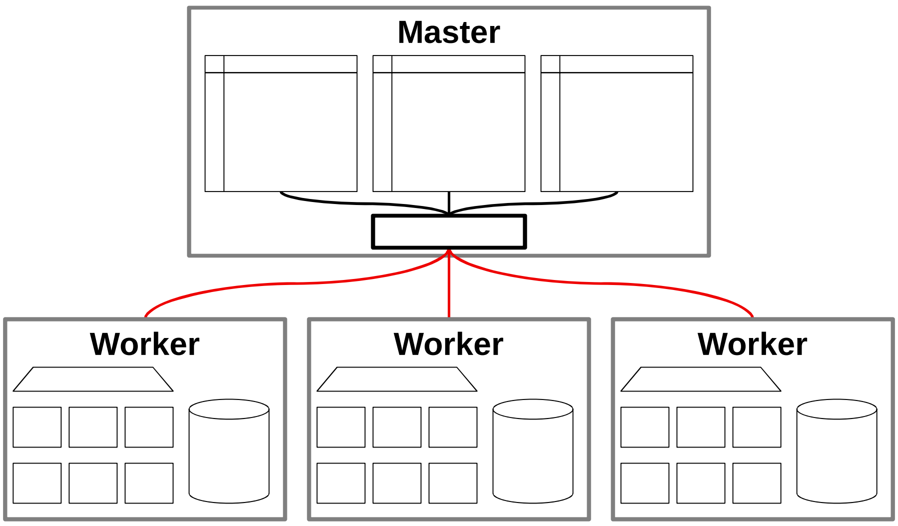
- Allow tasks to spawn other tasks



CIEL

- Execution engine for dynamic task graphs
- Supports various execution languages
  – Including *Skywriting* (later)
- Reliable execution on a distributed cluster
  – Client/master/worker fault tolerance (later)

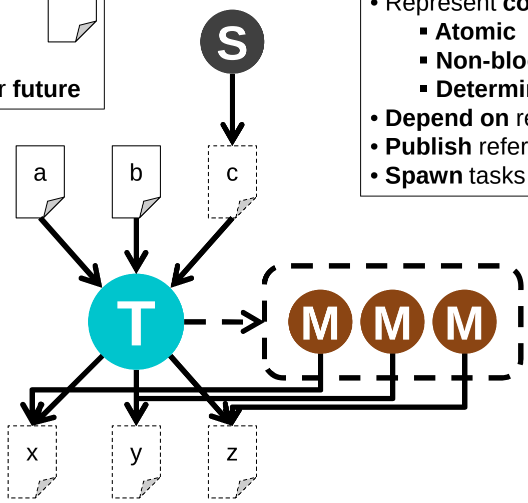
CIEL architecture



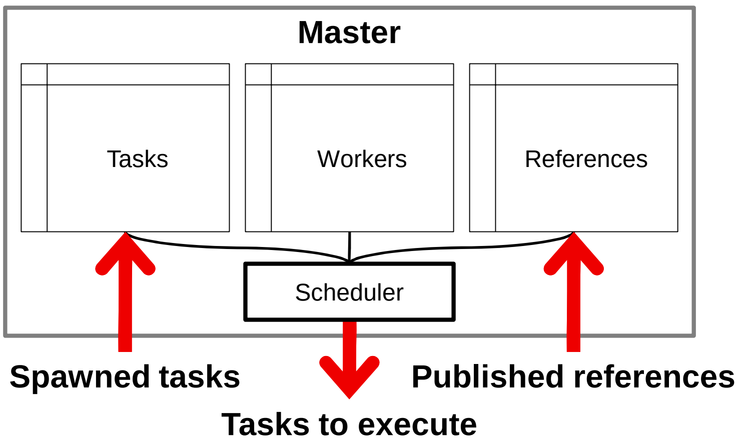
Tasks and references

References
- Name **objects**
- Have **GUIDs**
- Are **immutable**
- May be **concrete** or **future**

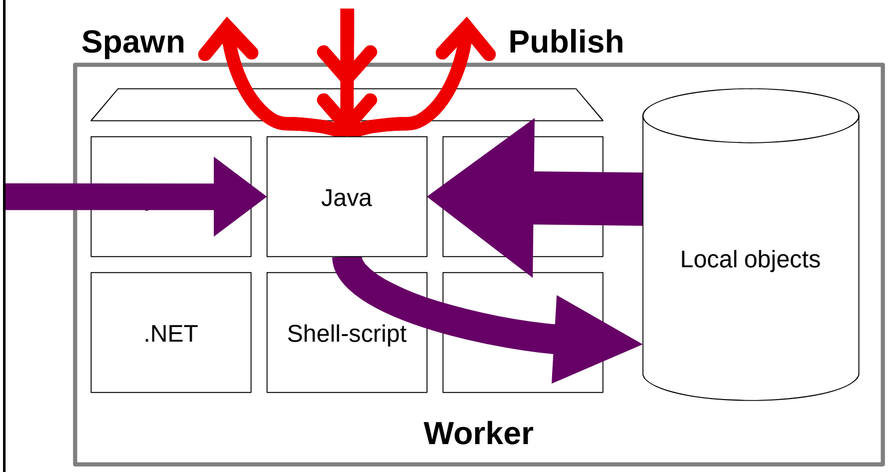
Tasks
- Represent **computation**
  - **Atomic**
  - **Non-blocking**
  - **Deterministic**
- **Depend on** references
- **Publish** references
- **Spawn** tasks



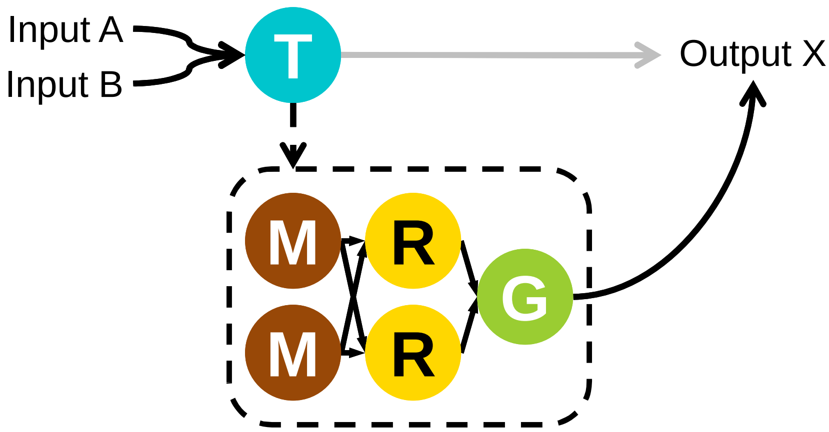
CIEL architecture



CIEL architecture



Defining dynamic task graphs



Skywriting

- Language for dynamic task graphs
  - Interpreted, dynamically-typed, C-like syntax
  - …including a `while` statement
- Runs end-to-end on CIEL
  - One script, one job
  - Stored-program model
  - Fault tolerance throughout execution

Skywriting in a nutshell

```
x = spawn(f);
y
retu
```

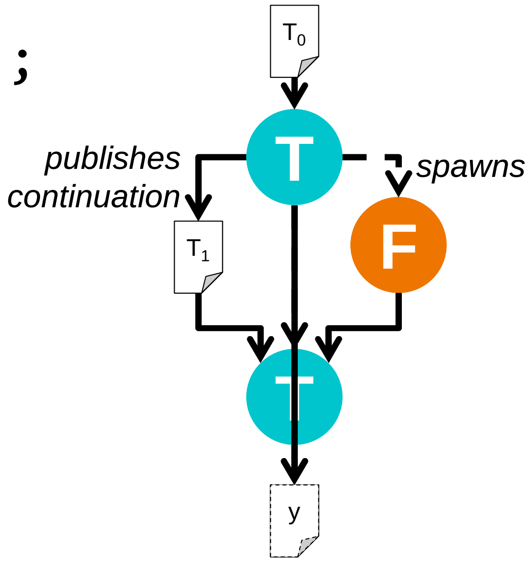
Future
referenc

Function
da

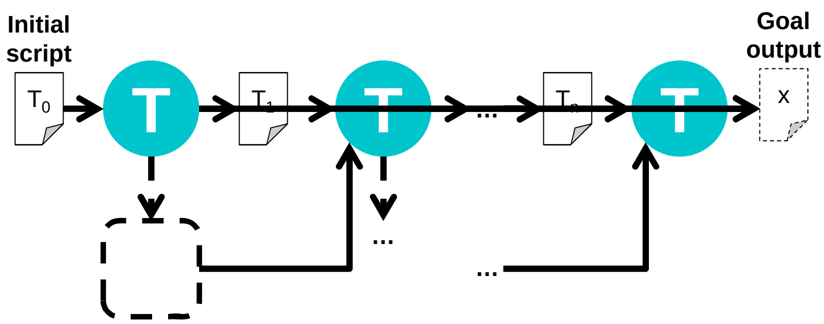
Dereference
operator

Blocking on futures

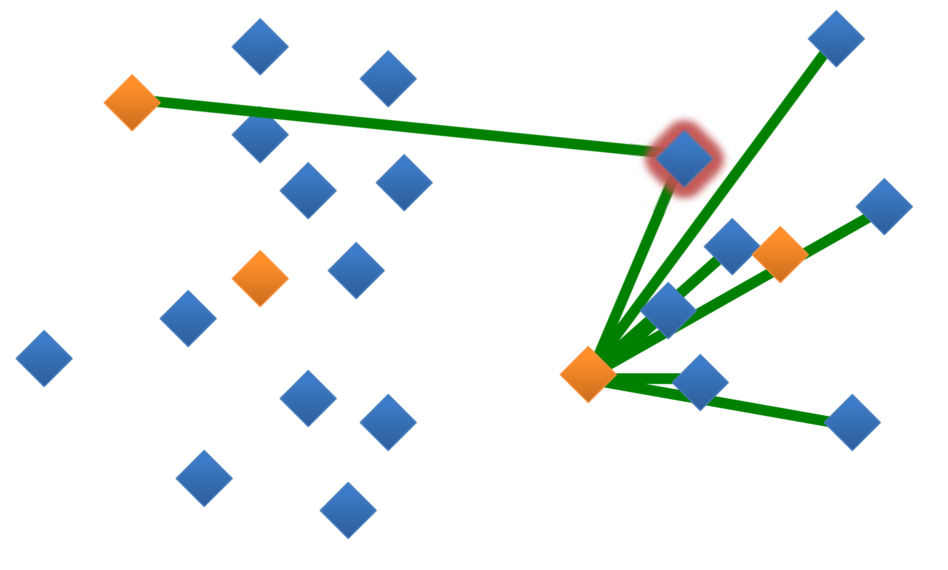
```
x = spawn(f);
y = *x;
return y;
```



Iteration through continuations

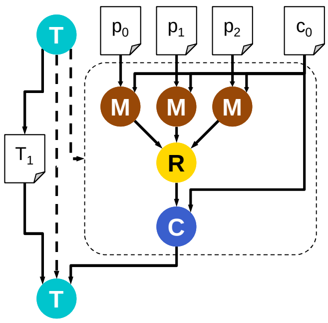


k-means clustering

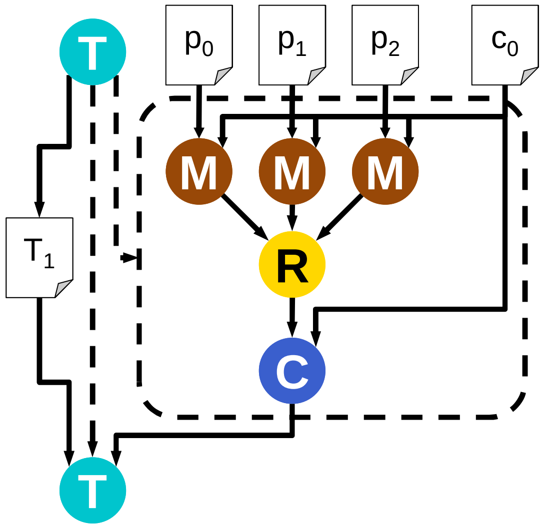


k-means in Skywriting

```
points = […]; curr = …;
do {
 sums = [];
 for (p in points) {
  sums += spawn(km_map, [p, curr]);
 }
 old = curr;
 curr = spawn(km_reduce, [sums]);
 done = spawn(is_converged, [curr, old]);
} while (!*done);
return curr;
```



k-means on CIEL

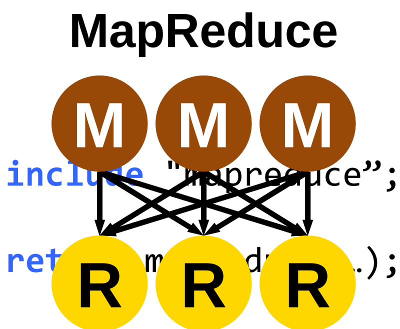


MapReduce in Skywriting

```
function apply(f, list) {
  outputs = [];
  for (i in range(len(list))) {
    outputs[i] = f(list[i]);
  }
  return outputs;
}

function shuffle(inputs, num_outputs) {
  outputs = [];
  for (i in range(num_outputs)) {
    outputs[i] = [];
    for (j in range(len(inputs))) {
      outputs[i][j] = inputs[j][i];
    }
  }
  return outputs;
}

function mapreduce(inputs, mapper, reducer, r) {
  map_outputs = apply(mapper, inputs);
  reduce_inputs = shuffle(map_outputs, r);
  reduce_outputs = apply(reducer, reduce_inputs);
  return reduce_outputs;
}
```



Reliable execution on CIEL

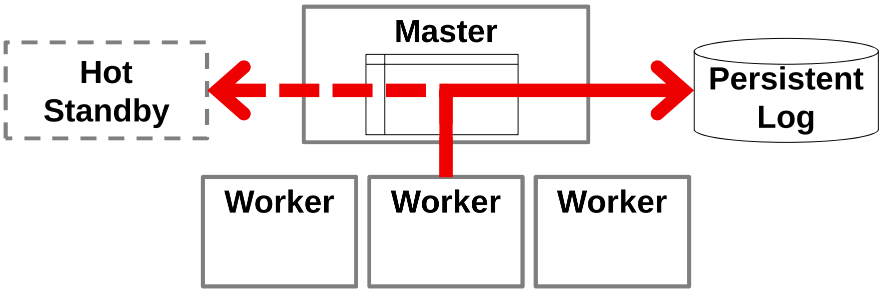
- Any participant in the computation can fail
- **Client** fault tolerance
  - Trivial due to whole-program execution
- **Worker** fault tolerance
  - Re-execute tasks as necessary
- **Master** fault tolerance

Task and reference naming

- Task (re-)execution must be **deterministic**
- In dynamic graph, how to choose names?
  - Deterministic function (SHA-1) of task inputs
- Lazy evaluation + deterministic naming
  - Task result memoization

Master fault tolerance

- Trivial version
  - Persistently store the root task for each job
- Better version



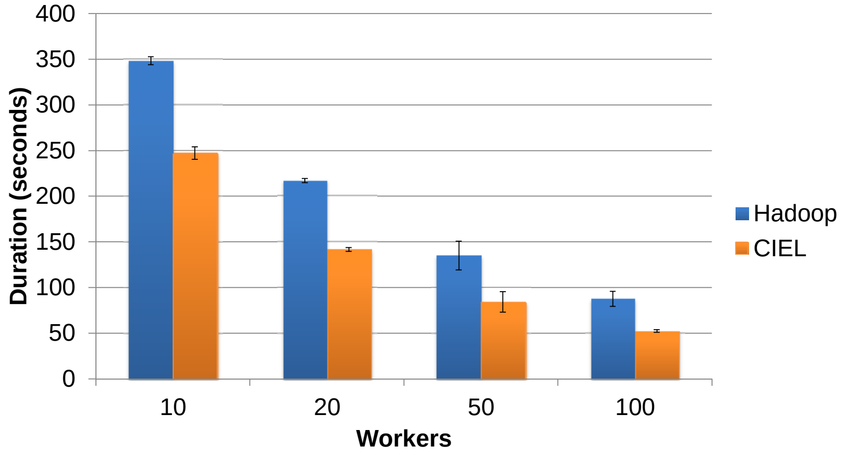
Implementation

- Implemented in ~8500 lines of Python
  - Plus executor code in Java, C, C#, …
- Client/server based on JSON-RPC/HTTP
- http://github.com/mrry/ciel

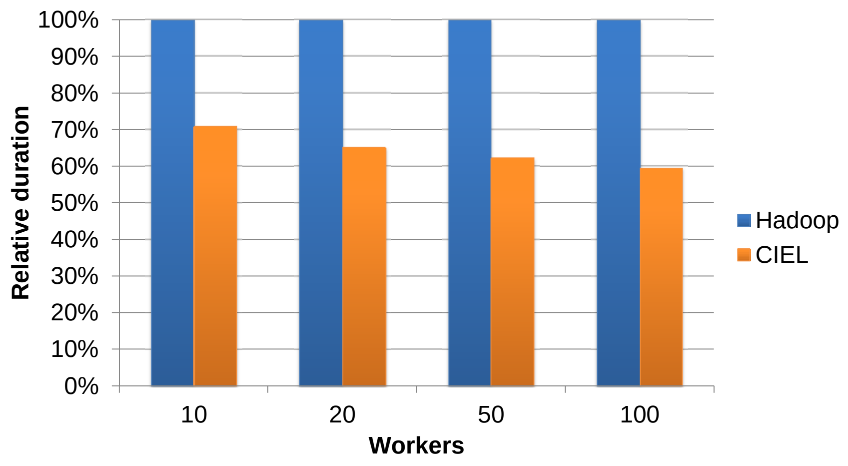
Applications

- Text-processing
  - Grep
  - Word count
- Clustering
  - *k*-means
- Link analysis
  - PageRank
- Linear algebra
  - Conjugate gradient
- Bioinformatics
  - Smith-Waterman
- Finance
  - Options pricing

Grep

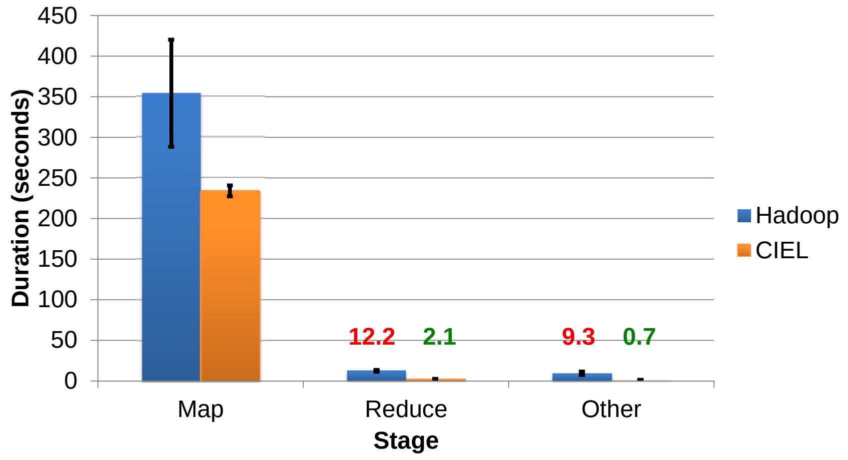


Grep



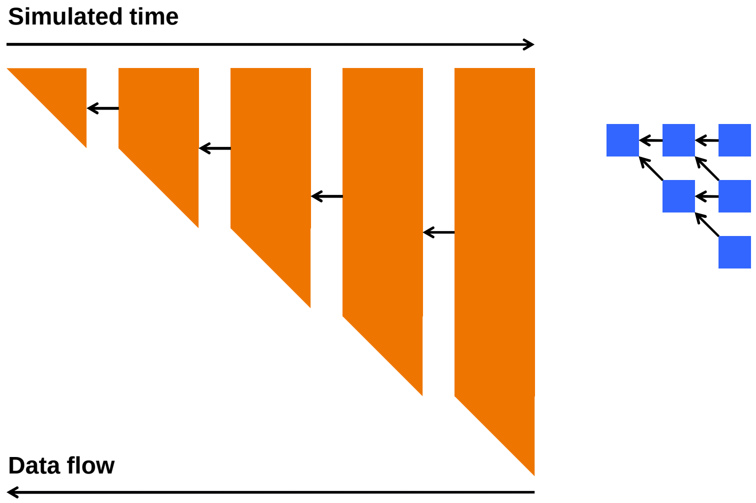
Relative duration
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
10
20
50
100
Workers
Hadoop
CIEL

k-means

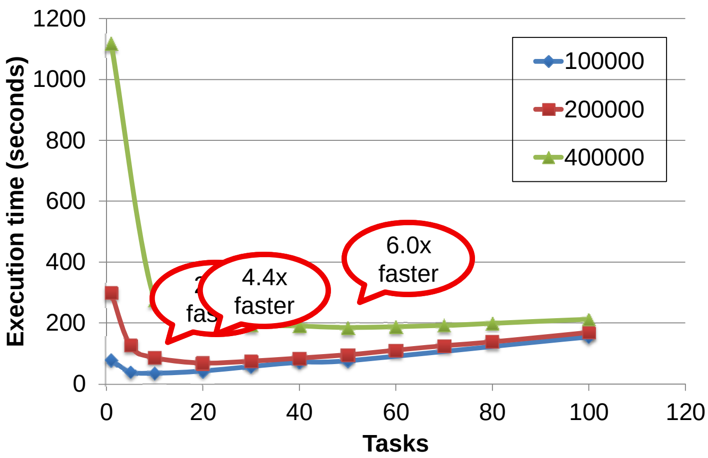


Duration (seconds)
450
400
350
300
250
200
150
100
50
0
12.2
2.1
9.3
0.7
Map
Reduce
Other
Stage
Hadoop
CIEL

Binomial options pricing



Binomial options pricing



Short-term future work

- More first-class languages on CIEL
  - Scala, Haskell, Java, Ocaml, …
- More platforms
  - Many-core and more exotic (e.g. Intel SCC)
- Hybrid multicore/distributed scheduling
  - Worker-local scheduler for lightweight tasks
  - Multi-scale concurrency interface

Longer-term vision

- Rumors of SMP's demise
  - Non-CC architectures (Beehive, SCC)
  - …or virtual machines in the cloud
  - …or both of the above
- How will we write applications in future?
  - Skywriting ≈ shell scripting

Conclusions

- Adding control-flow extends the class of programs that run on execution engines
- Skywriting lets you program in a simple imperative programming model
- CIEL achieves flexibility with competitive performance

http://www.cl.cam.ac.uk/netos/ciel/