UNIVERSITY OF CAMBRIDGE

CIEL: a universal execution engine for distributed data-flow computing

R244: Large-Scale Data Processing and Optimisation

Felix Jonathan Rocke

Murray, D. G., Schwarzkopf, M., Smowton, C., Smith, S., Madhavapeddy, A., & Hand, S. (2011). {CIEL}: A universal execution engine for distributed {Data-Flow} computing. In 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI 11).

Background

Using explicit message passing to run demanding programs on distributed systems

Development of execution engines for distributed computing to speed up development

Emergence of execution engines such as MapReduce and Dryad

Inefficient for iterative algorithms (often required in ML and optimisation tasks)

Development of CIEL and Skywriting

Design Goals of CIEL and Skywriting

CIEL (Execution Engine)

- Dynamic Control Flow
- Dynamic Task Dependencies
- Transparent Fault Tolerance
- Data Locality
- Transparent Scaling

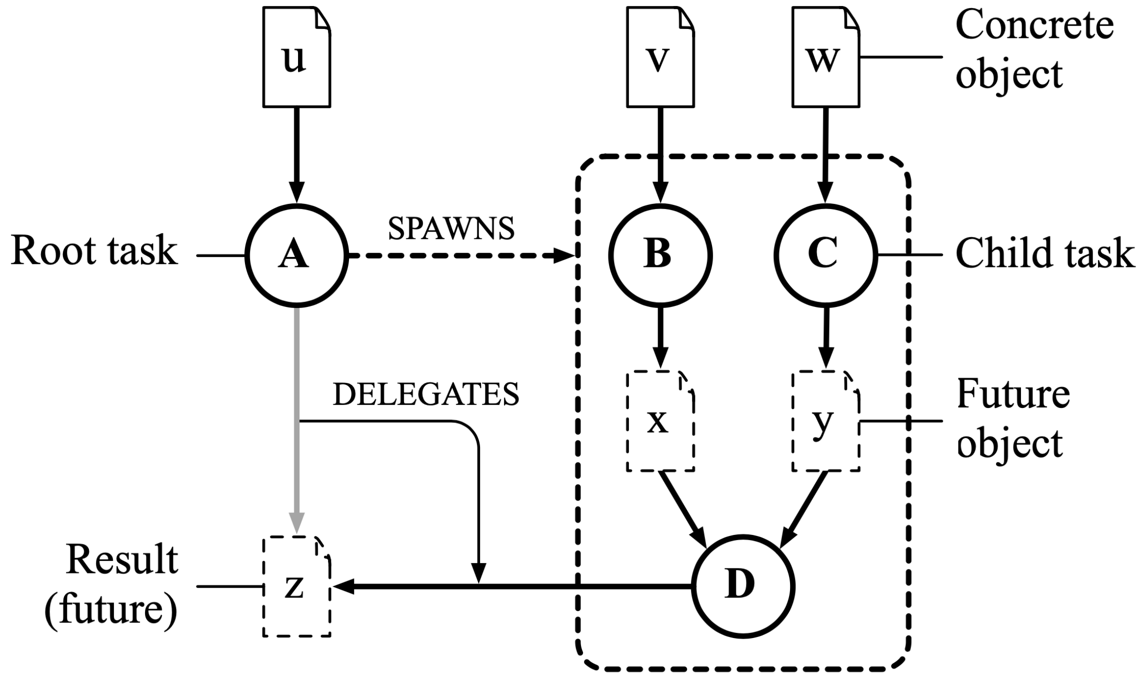
Skywriting (Scripting Language)

- Simplify expression of iterative & recursive task parallel algorithms
- Imperative syntax
- Functional syntax

Hide and handle the complexities of distributed computing and iterative algorithms

The Dynamic Task Graph

1. Root task spawns child tasks B & C
2. Child tasks receive concrete objects
3. Child tasks will produce future objects
4. Task D depends on the future objects
5. Task D promises result object z, of which the production was delegated from the root task to the children

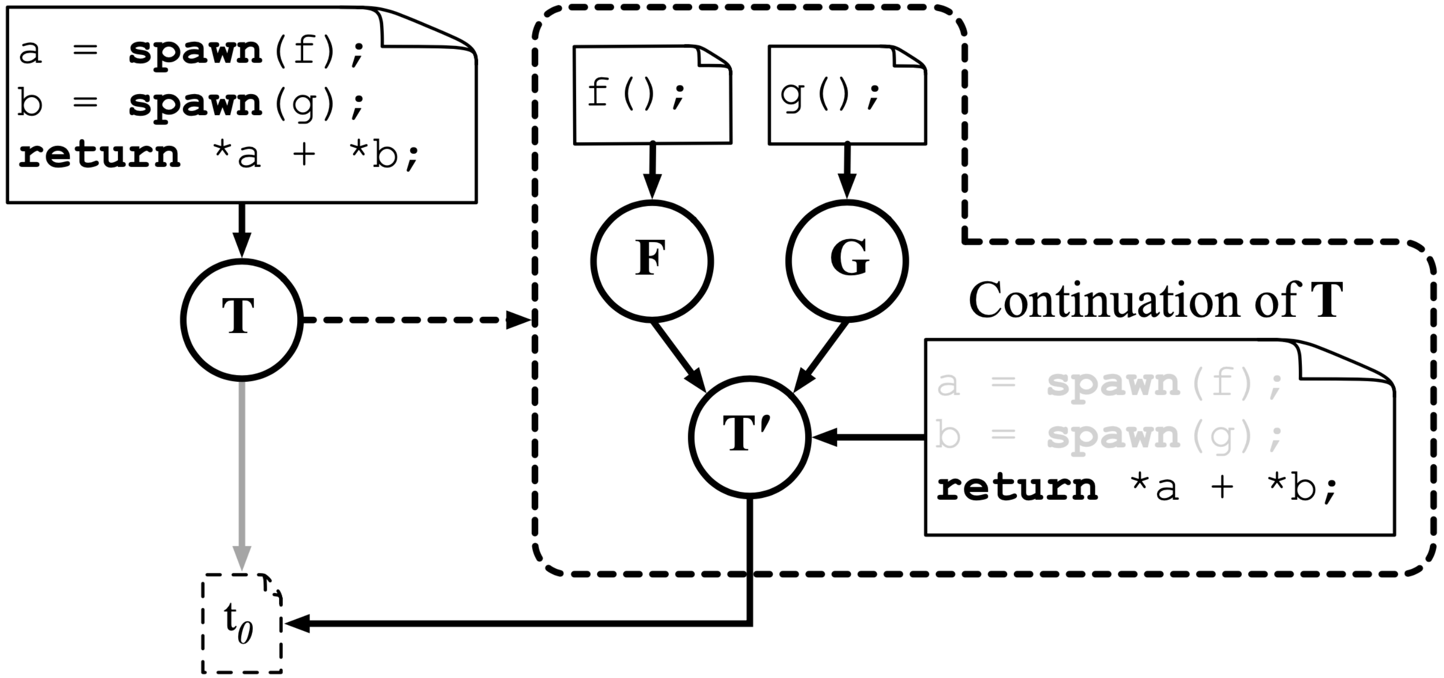


Dynamic Task Graph

Remember: Task graph is dynamically build to support data-dependent control flow

Example Creating Tasks Using Skywriting

1. Task T spawns Task F with function f
2. Task T spawns Task G with function g
3. Task T finishes and adds continuation task T’ to DAG
4. Tasks F and G return references
5. Task T’ dereferences a and b, i.e. loads their value into its context and returns the result of the addition



```
a = spawn(f);
b = spawn(g);
return *a + *b;
```

Spawning tasks and implicit task continuation due to dereferencing

Grep Benchmark CIEL and Hadoop

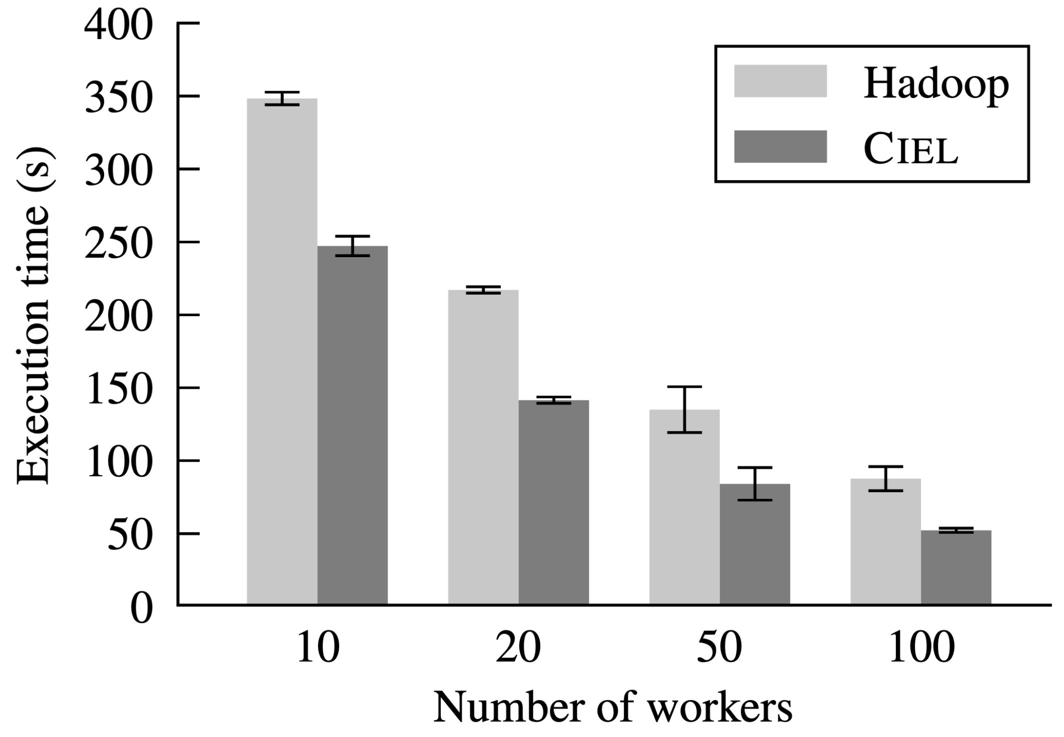
Task: Search 22.GB of English language Wikipedia for a three-character string

Consists of two MapReduce jobs:

- Parse input and emit matching strings
- Sort matches by frequency

Hadoop has a higher job overhead

- ➢ CIEL is, on average, 35% faster
- ➢ CIEL 29% faster with 10 workers
- ➢ CIEL 40% faster with 100 workers
- ➢ Relative performance improves with shorter jobs

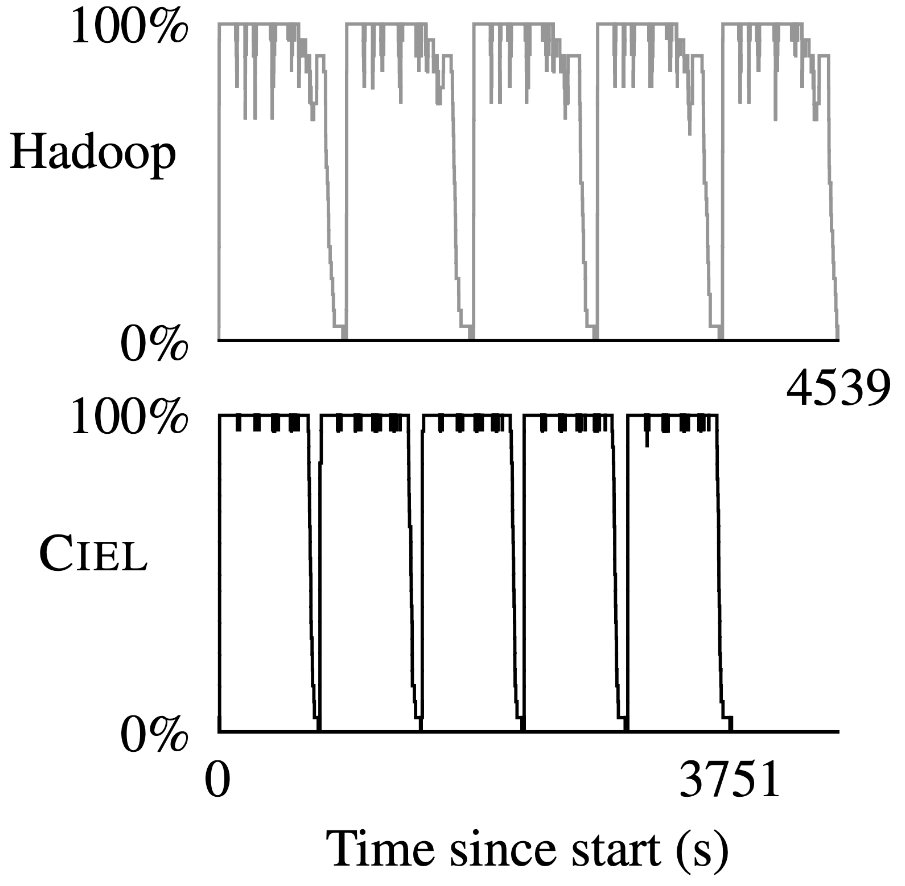


Grep execution time on Hadoop and CIEL

k-means Benchmark CIEL and Hadoop

Results:
- CIEL averages 89% ± 2% utilisation across all job sizes
- Hadoop achieves 84% for 100 tasks and only 59% utilisation for 20 tasks

Explanation:
- Overhead in Hadoops task scheduler
- CIEL has less fluctuation in task duration
  - Slow Hadoop tasks are not data-local, i.e. need to read data from another node
  - CIEL scheduler takes data locality into account



Cluster utilization for 5 iterations of 100 tasks

Pros & Cons / Discussion

Cons:

- Only Skywriting scripts can generate tasks, thus limiting usability
  - Rewriting of drivers for use with CIEL necessary
- No control over multiple cores in one machine inside of Skywriting
  - Developers need to spend more time on how their programs take advantage of the machines

Pros:

- Combines proven features from multiple distributed execution engines
- Provides robust and scalable performance for iterative algorithms
- Transparent fault tolerance, which allows recovery from machine failure
- Scheduling which takes memoisation and data-locality into consideration