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

Derek G. Murray, Malte Schwarzkopf, Christopher Smowton, Steven Smith, Anil Madhavapeddy, Steven Hand
University of Cambridge Computer Laboratory

Presented by Claire Coffey for R244

Motivations

- Other distributed execution engines (MapReduce, Dryad) built for processing large datasets
- Did not perform well for iterative algorithms
- Poor performance due to design: maximise throughput, not minimise job latency
- Latency increases when jobs are chained
- CIEL uses data-dependent control flow approach to combat
- Work created dynamically based on results of previous computations

Other Research Going On

- Also data-dependent control flow:
  - Google’s Pregel: executing graph algorithms, but only operates on single data set
- Solving iterative algorithm difficulties:
  - CGL-MapReduce: implementation of MapReduce, caches data across jobs
  - HaLoop: Hadoop extended
- Piccolo: programming model for data-parallel programming
  - Replaces reduce phase of MapReduce with partitioned key-value table

What’s Changed Since

- Techniques to aid in flexibility and performance:
  - Resilient Distributed Datasets
    - Distributed memory abstraction
    - Fault-tolerance in in-memory computations
    - Addresses inefficiencies in iterative algorithms and interactive data mining tools
  - Naiad
    - Distributed system, focuses on cyclic dataflow programs

What's Changed Since

- Developments using similar techniques, applied to machine learning:
  - TensorFlow
    - Also built to execute data flow graphs across cluster
    - Dataflow scheduler uses similar algorithm to CIEL
    - Interface and implementation for machine learning problems
  - RLGraph
    - Distributed execution for deep reinforcement learning problems

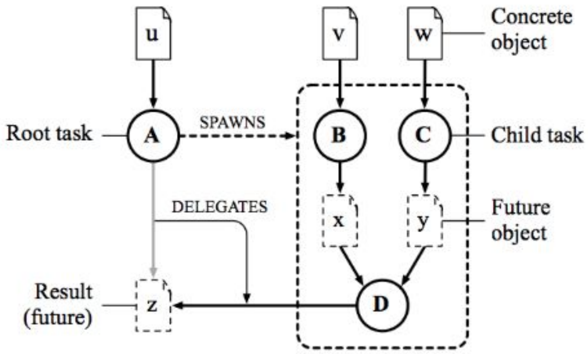
Problem to Solve

- MapReduce, Dryad, etc..., only perform well on some algorithms
- Struggle with iterative algorithms
- Iterative algorithms require more powerful execution engine
- Applications in machine learning and optimisation

Key Ideas: Dynamic Task Graph

- Executes programs
- Arbitrary data-dependent control flow
- CIEL job represented as Dynamic Task Graph (DTG)
    - 3 key primitives interact to form DTG:
        - **Objects**
        - **References**
        - **Tasks**
- Execution data-centric, each job produces 1+ objects
- DTG stores relations between tasks and objects

Key Ideas: Dynamic Task Graph



(a) Dynamic task graph

Task ID	Dependencies	Expected outputs
A	{ u }	~~z~~
B	{ v }	x
C	{ w }	y
D	{ x, y }	z

Object ID	Produced by	Locations
u	–	{ `host19`, `host85` }
v	–	{ `host21`, `host23` }
w	–	{ `host22`, `host57` }
x	**B**	∅
y	**C**	∅
z	~~**A**~~ **D**	∅

(b) Task and object tables

Example DTG with corresponding task and object tables

Figure 2: A CIEL job is represented by a dynamic task graph, which contains tasks and objects (§3.1). In this example, root task **A** spawns tasks **B**, **C** and **D**, and delegates the production of its result to **D**. Internally, CIEL uses task and object tables to represent the graph (§3.3).

Source: D. Murray et al.: CIEL: a universal execution engine for distributed data-flow computing

Key Ideas: Skywriting

- Language runs on top of CIEL, designed for data-centric computations
- Expresses arbitrary data-dependent control flow with loops and recursive functions, can create tasks
- Key features:
  - **ref(url)**
  - **spawn(f, [arg,...])**
  - **exec(executor, args, n)**
  - **spawn exec(executor, args, n)**
  - **Dereference operator (-*)**

Key Ideas: Skywriting

Example iterative computation in Skywriting

input _data - list of *n* input chunks

curr - initialised to list of *n* partial results

Source: D. Murray et al.: CIEL: a universal execution engine for distributed data-flow computing

```
function process_chunk(chunk, prev_result) {
  // Execute native code for chunk processing.
  // Returns a reference to a partial result.
  return spawn_exec(...);
}

function is_converged(curr_result, prev_result) {
  // Execute native code for convergence test.
  // Returns a reference to a boolean.
  return spawn_exec(...)[0];
}

input_data = [ref("ciel://host137/chunk0"),
              ref("ciel://host223/chunk1"),
              ...];
curr = ...; // Initial guess at the result.

do {
  prev = curr;
  curr = [];
  for (chunk in input_data) {
    curr += process_chunk(chunk, prev);
  }
} while (!*is_converged(curr, prev));

return curr;
```

What They Did: Implementation

- Implemented CIEL distributed execution engine and Skywriting language

- Goal of development to support a more powerful computation model than existing distributed execution engines

- Important not to sacrifice performance

What They Did: Evaluation

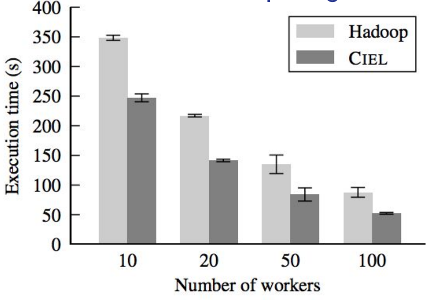
- Evaluated success by:
  - Comparison to Hadoop (popular MapReduce system)
  - Benefits when executing iterative algorithms
  - Overheads on compute intensive tasks
  - Effect of master failure on performance
- Multiple experiments:
  - **Grep** search compared to Hadoop
  - **K-means** clustering compared to Hadoop
  - **Binomial Options Pricing**: dynamic programming algorithm, difficult to parallelise
  - **Smith-Waterman** sequence alignment algorithm: dynamic programming algorithm, difficult to parallelise
  - **Fault Tolerance**: master fail-over induced during iterative computation

Evaluation Results

- **Grep**: averaged across runs, CIEL outperforms Hadoop by 35%
- **K-means**:
  - CIEL faster than Hadoop on all job sizes
  - Task duration: Hadoop distribution bimodal; 64% “fast” tasks, 36% “slow” tasks; all CIEL tasks “fast”

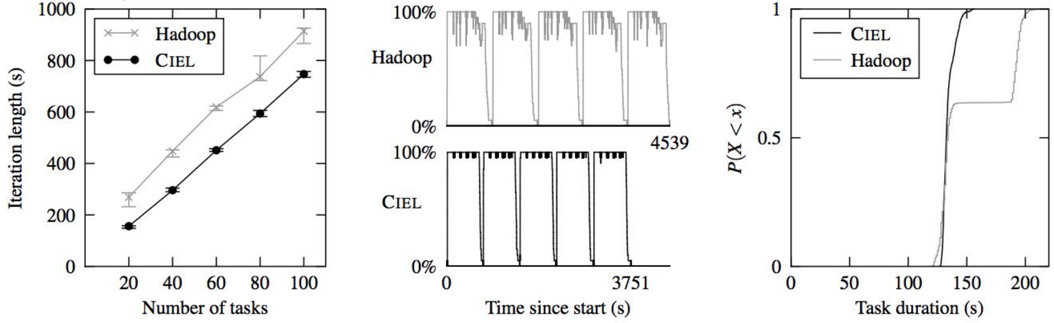
Grep results

Source: D. Murray et al.: CIEL: a universal execution engine for distributed data-flow computing



K-Means results

Source: D. Murray et al.: CIEL: a universal execution engine for distributed data-flow computing



(a) Iteration length (b) Cluster utilisation (100 tasks) (c) Map task distribution

Evaluation Results

- **Smith-Waterman**:
    - Does not perform well overall
    - Matrix size 30x30 results satisfactory
    - Otherwise cannot achieve full utilisation (smaller and larger sizes)
- **Binomial Options Pricing**:
    - Maximum speedup increases as problem size grows - amount of independent work in each task grows
    - After maximum, speedup decreases - small tasks suffer from constant per-task overhead
- **Fault tolerance**:
    - Between failure of master and resumption, 7.7 seconds elapse
    - Utilisation during second iteration worse - tasks must be replayed
    - Back to full utilisation by 3rd iteration
    - Overall job execution time increases

Strengths and Agreements

- Good solution for iterative algorithm execution
  - Alternative engines couldn't handle this
  - Useful in machine learning and optimisation
  - Real problem
- Skywriting - easy expression of algorithms
- Evaluation looks at results in-depth for algorithm comparisons
  - e.g. k-means looks at the iteration length, cluster utilisation and map task distribution

Weaknesses and Disagreements

- No control over data caching
    - If configurable could exploit data for faster performance
- Programs must be rewritten in Skywriting - only Skywriting programs can create new tasks
    - Annoying, puts pressure on runtime for interpreted code
- Scaling challenges
    - Multiple cores not used effectively - each executor has full use of machine, limiting efficiency if program is sequential and multiple cores available
- Fault tolerance slow
- For dynamic programming algorithms, no comparison to alternative engines

Key Takeaways

- Satisfies same features as existing distributed execution engines
- Additionally, efficient execution of iterative algorithms
- Skywriting provides simple way to express iterative algorithms in imperative way, fault tolerant
- CIEL performs well in comparison to Hadoop on iterative algorithms
- Fault tolerance successful, quite slow
- Mixed success on dynamic programming algorithms, but no comparison to alternatives

Impact

- Well-received, 287 citations
  - Good/relevant/interesting
- Authors did not publish more on CIEL
  - Suggests not built upon by authors
- Most cite as related and relevant system. Propose either:
  - Similar system for different problem, e.g Naiad - cyclic data flows
  - Or applied to specific problem, e.g. TensorFlow - similar scheduling algorithm applied to machine learning

Questions?