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

INTRODUCTION

- Background Influences
- What is CIEL?
- Features
- Skywriting
- Evaluation
- Conclusions

BACKGROUND INFLUENCES

- Map-Reduce/Hadoop
- Dryad
- Pregel
- Piccolo

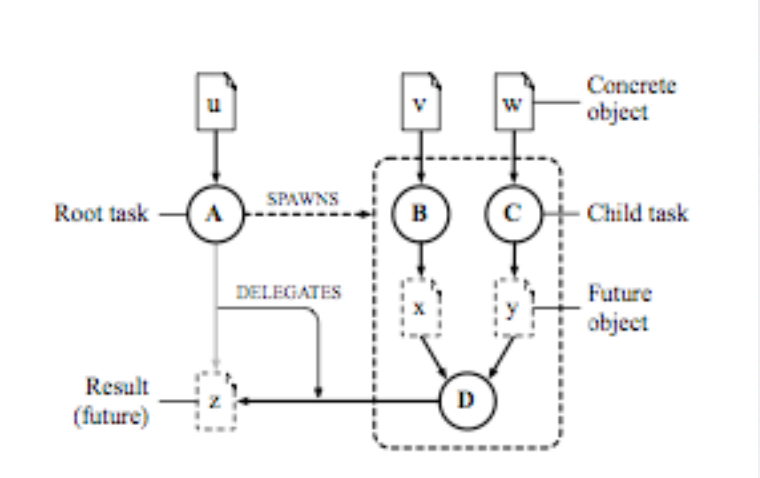
WHAT IS CIEL?

- Universal data-centric distributed execution engine
- Designed for large dataset, coarse-grained parallelism
- Based on data-dependent dynamic control flow
- Uses 3 primitives - objects, references and tasks
- Primary Goal is to produce object output

FEATURES

- **Dynamic task graphs**
- System architecture
- Deterministic naming & Memoisation
- Fault tolerance
- Streaming

DYNAMIC TASK GRAPHS



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

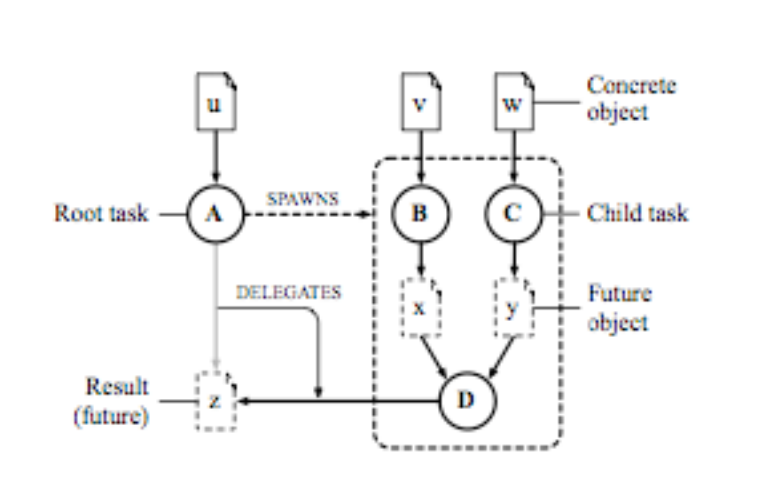
Object ID	Produced by	Locations
u	–	{ host19, host85 }
v	–	{ host21, host23 }
w	–	{ host22, host57 }
x	**B**	∅
y	**C**	∅
z	~~**A**~~ **D**	∅

(b) Task and object tables

Objects

- Unstructured finite-length sequence of bytes
- Unique name
- Immutable when written

DYNAMIC TASK GRAPHS



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

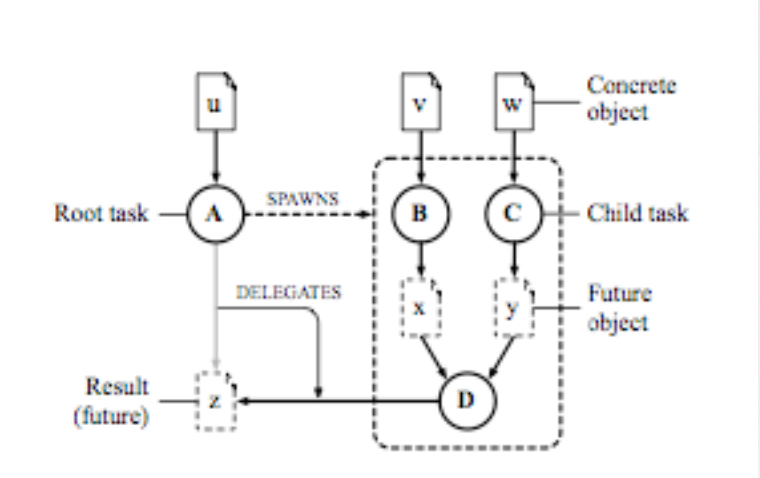
Object ID	Produced by	Locations
u	–	{ host19, host85 }
v	–	{ host21, host23 }
w	–	{ host22, host57 }
x	**B**	∅
y	**C**	∅
z	~~**A**~~ **D**	∅

(b) Task and object tables

References

- Comprises name and set of locations where object is stored
- Can be a future reference to object yet produced

DYNAMIC TASK GRAPHS



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

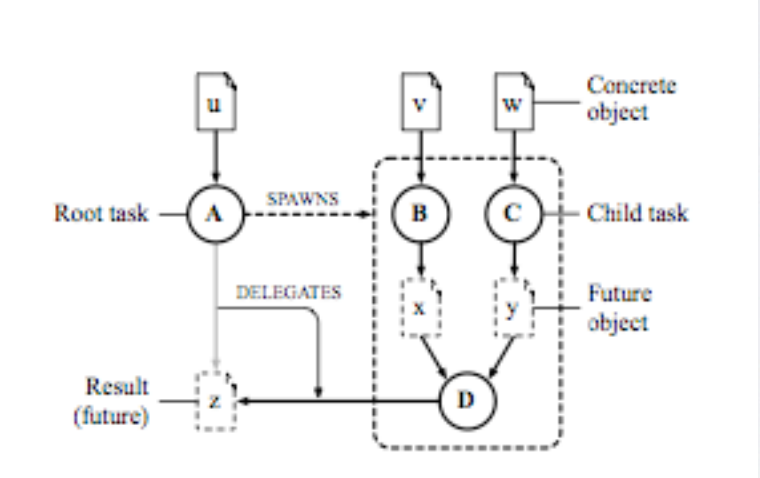
Object ID	Produced by	Locations
u	–	{ host19, host85 }
v	–	{ host21, host23 }
w	–	{ host22, host57 }
x	**B**	∅
y	**C**	∅
z	~~**A**~~ **D**	∅

(b) Task and object tables

Tasks

- Non-blocking atomic computation
- Has one or more dependencies - represented as references
- Includes special object that specifies the behaviour of the task
- Two externally-observable behaviours - publish objects and spawn new tasks

DYNAMIC TASK GRAPHS



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

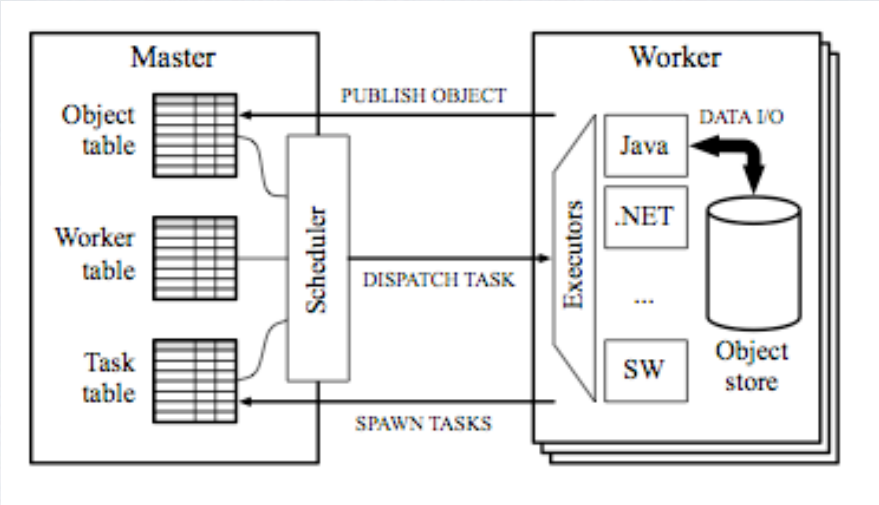
Object Evaluation

- Role = evaluate one or more objects corresponding to job outputs
- Job can be specified as single root task with only concrete dependencies
- Two natural strategies - Eager and Lazy evaluation

FEATURES

- Dynamic task graphs
- **System architecture**
- Deterministic naming & Memoisation
- Fault tolerance
- Streaming

SYSTEM ARCHITECTURE



- Single master coordinating end-to-end execution of jobs
- Several workers are used for execution of individual tasks
- DTG maintained by master in object and task table
- Master Scheduler (multiple queue based) responsible for making progress in CIEL computation
- Executor = generic component that prepares input data for consumption

FEATURES

- Dynamic task graphs
- System architecture
- **Deterministic naming & Memoisation**
- Fault tolerance
- Streaming

FEATURES

- Dynamic task graphs
- System architecture
- Deterministic naming & Memoisation
- **Fault tolerance**
- Streaming

FEATURES

- Dynamic task graphs
- System architecture
- Deterministic naming & Memoisation
- Fault tolerance
- **Streaming**

SKYWRITING

- Key Features - ref, spawn, exec., spawn.exec, the dereference operator
- Tasks - key feature = ability to spawn new tasks in the middle of jobs
- Data-dependent control flow

EVALUATION

- Grep
- *k*-means
- Smith-Waterman
- Binomial options pricing
- Fault-tolerance

CONCLUSIONS

- Superset of features of existing distributed engines
- Skywriting
- Flexibility - Supports MapReduce job or Dryad graph
- System-wide fault tolerance
- Streaming
- Memoisation

THANKS

- Any Questions?