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

Derek Murray, Malte Schwarzkopf, Christopher Smowton, Steven Smith, Anil Madhavapeddy and Steven Hand

Bogdan-Alexandru Matican

University of Cambridge

February 5, 2013

Table of contents

Main considerations

- distributed data-flow computing
- task dependencies
- dynamic coordination

Bonus: transparency (fault tolerance, scaling, locality)

Feature	MapReduce [2, 18]	Dryad [26]	Pregel [28]	Iterative MR [12, 21]	Piccolo [34]	CIEL
Dynamic control flow	✗	✗	✓	✓	✓	✓
Task dependencies	Fixed (2-stage)	Fixed (DAG)	Fixed (BSP)	Fixed (2-stage)	Fixed (1-stage)	Dynamic
Fault tolerance	Transparent	Transparent	Transparent	✗	Checkpoint	Transparent
Data locality	✓	✓	✓	✓	✓	✓
Transparent scaling	✓	✓	✓	✓	✗	✓

Figure : Features of distributed execution engines.

Introduction

The system is primarily focused around the following:

Data objects and references to them

Processing tasks (input and output references)

Coordination dynamic task graph

Managing the graph

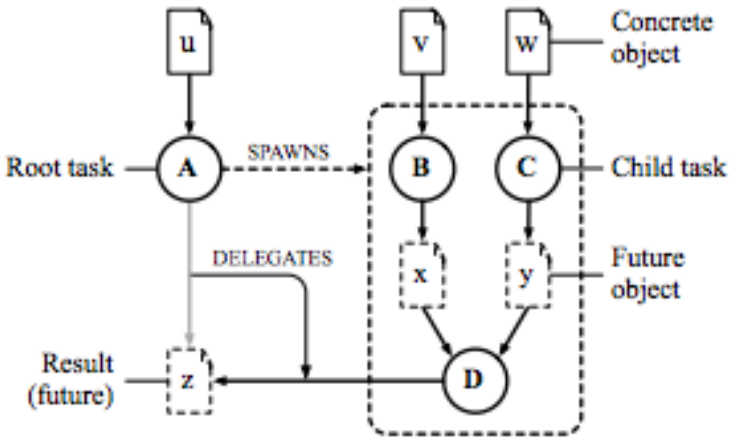
Two main rules for dependencies:

- **Input** depend on concrete or future references
- **Output** publish reference **OR** spawn child

Two main evaluation styles:

- **Eager** start with concrete tasks and continue
- **Lazy** start from root moving recursively down

Example state



(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

Figure : A CIEL job example.

Architecture

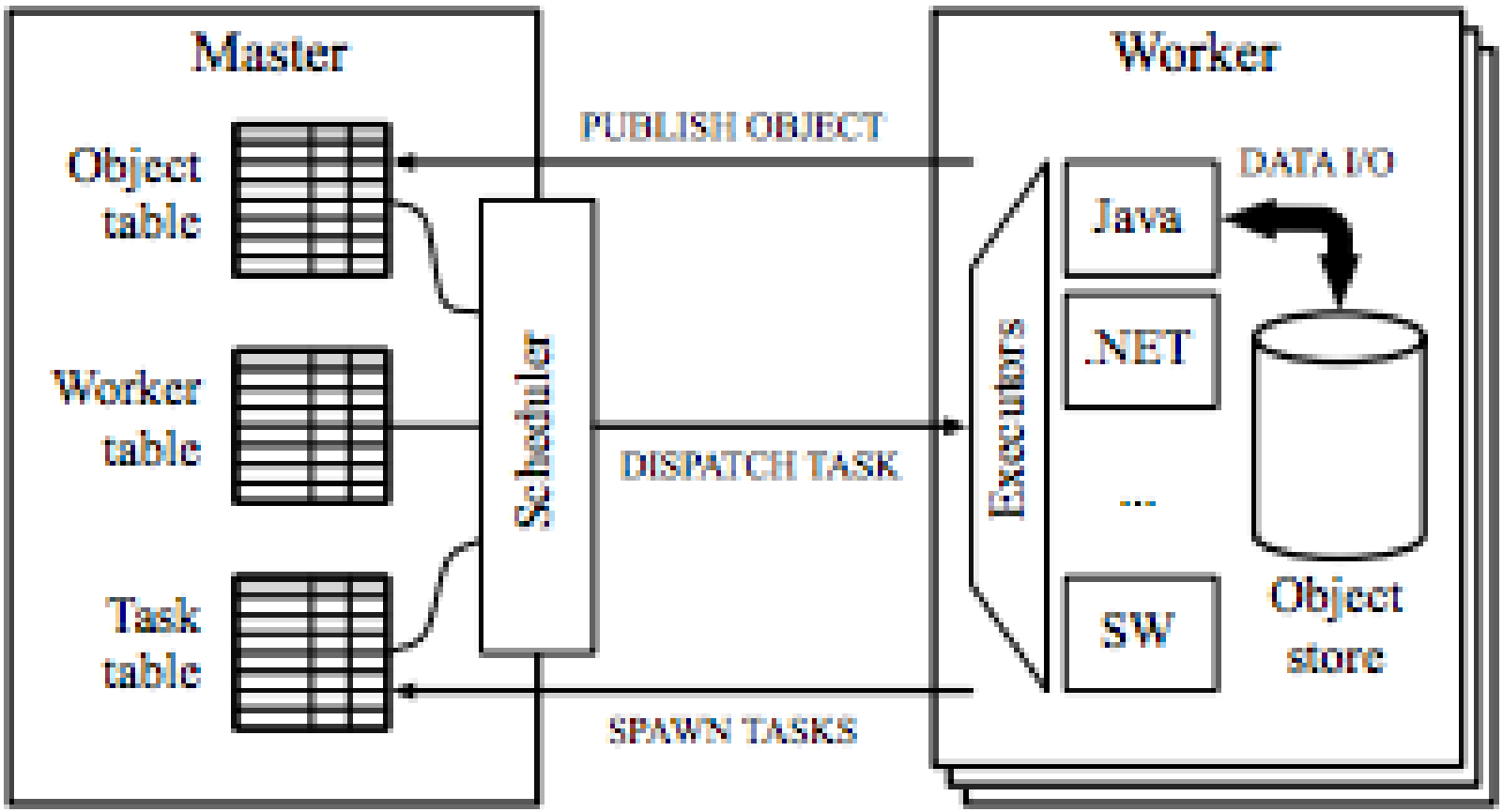


Figure : Cluster architecture

Introduction

A couple of important primitives:

- *spawn* – parallel task
- *exec* – synchronous executor
- *dereference* – load reference in context

Handling tasks

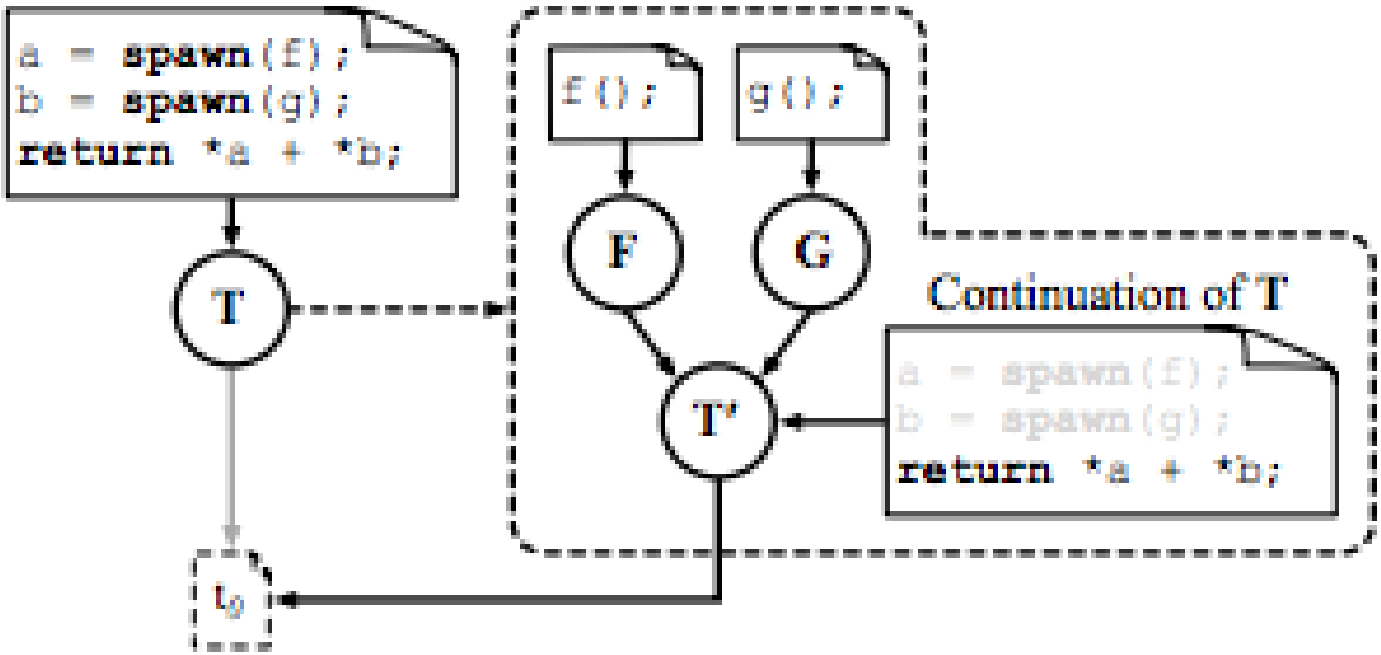


Figure : Task creation example.

Details

- dereferencing – data / coordination space
- naming and memoisation
- fault tolerance (client / worker / master)
- streaming

Contributions

- system with broader computational model
- dynamic task dependency handling
- transparent distribution and scheduling

Critique and questions

- is Skywriting as a language necessary?
- worker fault tolerance – replication?
- deterministic, terminating computation?
- homogenous machines in cluster?