Scripting the cloud with Skywriting

Derek G. Murray  Steven Hand
University of Cambridge
A universal model?

MapReduce
A universal model?

MapReduce
A universal model!
Move computation to the data

```java
Driver program
submitJob();
```

Code

Results
while (!converged)
    do work in parallel;
Iterative algorithm

Driver program

while (...) submitJob();
Iterative algorithm
```
while (...) 
  doStuff();
```
Skywriting

- JavaScript-like job specification language
  - Supports functional programming
  - Data-dependent control flow

- Distributed execution engine
  - Locality-based scheduling
  - Fault tolerance
  - Thread migration
function \( f(x) \) { return \( x + 1 \); }

\( \text{res1} = \text{spawn}(f, [42]); \)
function $f(x)$ { \text{return} \ x + 1; \}

function $g(y)$ { \text{...} \}

res1 = \text{spawn}(f, \ [42]);
res2 = \text{spawn}(g, \ [\text{res1}]);

res1 and res2 are future references
points = [...]; // List of partitions
w = ...; // Random initial value
for (i in range(0, ITERATIONS)) {
    w_old = w;
    results = [];
    for (part in points) {
        results += spawn(log_reg, [part, w_old]);
    }
    w = spawn(update, [w_old, results]);
}
Logistic regression

points = [...]; // List of partitions
w = ...;       // Random initial value

do {
    w_old = w;
    results = [];
    for (part in points) {
        results += spawn(log_reg, [part, w_old]);
    }
    w = spawn(update, [w_old, results]);
    done = spawn(converged, [w_old, w]);
} while (!*done);
Logistic regression

points = [...]; // List of partitions
w = ...;       // Random initial value

do {
    w_old = w;
    results = [];
    for (part in points) {
        results += spawn(log_reg, [part, w_old]);
    }
}
while (!*done);

*-operator dereferences (forces) a future
Implementation status

• Implemented in 4000 lines of Python
  – Also: Java, C and .NET bindings
• Many additional features
  – Native code execution
  – Introspection
  – Conditional synchronisation
• Available as open-source
  – http://github.com/mrry/skywriting
Job creation overhead

![Graph showing job creation overhead with overhead on the y-axis and number of workers on the x-axis. Two lines represent Hadoop and Skywriting, with Hadoop having a greater overhead than Skywriting for all numbers of workers.]
Future directions

• Multiple-scale parallel computing
  – Multiple cores, machines and clouds

• Streaming computations
  – Piping high-bandwidth data between tasks

• Better language integration
  – Hosted Skywriting on CLR or JVM
Conclusions

- Turing-complete programming language for distributed computation
- Runs real jobs with low overhead
- Lots more still to do!
Questions?

• Email
  – Derek.Murray@cl.cam.ac.uk

• Project website
  – http://www.cl.cam.ac.uk/netos/skywriting/