Toward a Common Model for Highly Concurrent Applications

Douglas Thain
University of Notre Dame

MTAGS Workshop
17 November 2013
Overview

• Experience with Concurrent Applications
  – Makeflow, Weaver, Work Queue

• Thesis: Convergence of Models
  – Declarative Language
  – Directed Graphs of Tasks and Data
  – Shared Nothing Architecture

• Open Problems
  – Transaction Granularity
  – Where to Parallelize?
  – Resource Management

• Concluding Thoughts
The Cooperative Computing Lab
University of Notre Dame

http://www.nd.edu/~ccl
The Cooperative Computing Lab

• We *collaborate with people* who have large scale computing problems in science, engineering, and other fields.

• We *operate computer systems* on the O(10,000) cores: clusters, clouds, grids.

• We *conduct computer science* research in the context of real people and problems.

• We *release open source software* for large scale distributed computing.

http://www.nd.edu/~ccl
Our Collaborators

AGTCCGTACGATGCTATTAGCGAGCGTGA...
Good News: Computing is Plentiful
CPU Utilization for the Last Week

404855  (51%) CPU-Hours Unused
328960  (41%) CPU-Hours Used by Condor
58935   (7%) CPU-Hours Used by Owner
792750  (100%) CPU-Hours Total
Superclusters by the Hour

$1,279-per-hour, 30,000-core cluster built on Amazon EC2 cloud

By Jon Brodkin | Published a day ago

The Bad News: It is inconvenient.
End User Challenges

• System Properties:
  – Wildly varying resource availability.
  – Heterogeneous resources.
  – Unpredictable preemption.
  – Unexpected resource limits.

• User Considerations:
  – Jobs can’t run for too long... but, they can’t run too quickly, either!
  – I/O operations must be carefully matched to the capacity of clients, servers, and networks.
  – Users often do not even have access to the necessary information to make good choices!
I have a standard, debugged, trusted application that runs on my laptop.

A toy problem completes in one hour.  
A real problem will take a month (I think.)

Can I get a single result faster?  
Can I get more results in the same time?

Last year,  
I heard about this grid thing.

This year,  
I heard about this cloud thing.

*What do I do next?*
Our Philosophy:

- Harness all the resources that are available: desktops, clusters, clouds, and grids.
- Make it easy to scale up from one desktop to national scale infrastructure.
- Provide familiar interfaces that make it easy to connect existing apps together.
- Allow portability across operating systems, storage systems, middleware...
- Make simple things easy, and complex things possible.
- *No special privileges required.*
An Old Idea: Makefiles

part1 part2 part3: input.data split.py
./split.py input.data

out1: part1 mysim.exe
./mysim.exe part1 >out1

out2: part2 mysim.exe
./mysim.exe part2 >out2

out3: part3 mysim.exe
./mysim.exe part3 >out3

result: out1 out2 out3 join.py
./join.py out1 out2 out3 > result
Makeflow = Make + Workflow

- Provides portability across batch systems.
- Enable parallelism (but not too much!)
- Fault tolerance at multiple scales.
- Data and resource management.

http://www.nd.edu/~ccl/software/makeflow
Makeflow Applications
Example: Biocompute Portal

- Generate Makefile
- Progress Bar
- Transaction Log
- Makeflow
- Update Status
- Submit Tasks
- Condor Pool

BLAST
SSAHA
SHRIMP
EST
MAKER
...
Generating Workflows with Weaver

db = SQLDataSet('db', 'biometrics', 'irises');
irises = Query(db, color=='Blue')

iris_to_bit = SimpleFunction('convert_iris_to_template')
compare_bits = SimpleFunction('compare_iris_templates')

bits = Map(iris_to_bit, irises)
AllPairs(compare_bits, bits, bits, bits, output='scores.txt')
Weaver + Makeflow + Batch System

• A good starting point:
  – Simple representation is easy to pick up.
  – Value provided by DAG analysis tools.
  – Easy to move apps between batch systems.

• But, the shared filesystem remains a problem.
  – Relaxed consistency confuses the coordinator.
  – Too easy for Makeflow to overload the FS.

• And the batch system was designed for large jobs.
  – Nobody likes seeing 1M entries in qstat.
  – 30-second rule applies to most batch systems
Work Queue System

1000s of workers dispatched to clusters, clouds, and grids

Work Queue Program
C / Python / Perl

Work Queue Library

put P.exe
put in.txt
exec P.exe <in.txt >out.txt
get out.txt

http://www.nd.edu/~ccl/software/workqueue
Makefile + Work Queue

- Makefile
- Makeflow
- Local Files and Programs
- submit tasks
- Private Cluster
- Shared SGE Cluster
- Campus Condor Pool
- Public Cloud Provider
- ssh

Hundreds of Workers in a Personal Cloud:
- sge_submit_workers
- condor_submit_workers
Managing Your Workforce

Master A

Master B

Master C

Condor Pool

Torque Pool

Submits new workers.
Restarts failed workers.
Removes unneeded workers.

work_queue_pool –T condor

work_queue_pool –T torque
#include "work_queue.h"

while( not done ) {
    while (more work ready) {
        task = work_queue_task_create();
        // add some details to the task
        work_queue_submit(queue, task);
    }

    task = work_queue_wait(queue);
    // process the completed task
}

http://www.nd.edu/~ccl/software/workqueue
Adaptive Weighted Ensemble

Proteins fold into a number of distinctive states, each of which affects its function in the organism.

How common is each state?
How does the protein transition between states?
How common are those transitions?
AWE Using Work Queue

Simplified Algorithm:

– Submit N short simulations in various states.
– Wait for them to finish.
– When done, record all state transitions.
– If too many are in one state, redistribute them.
– Stop if enough data has been collected.
– Continue back at step 2.
AWE on Clusters, Clouds, and Grids
New Pathway Found!

Credit: Joint work in progress with Badi Abdul-Wahid, Dinesh Rajan, Haoyun Feng, Jesus Izaguirre, and Eric Darve.
Software as a Social Lever

• User and app accustomed to a particular system with standalone executables.
• Introduce Makeflow as an aid for expression, debugging, performance monitoring.
• When ready, use Makeflow + Work Queue to gain more direct control of I/O operations on the existing cluster.
• When ready, deploy Work Queue to multiple systems across the wide area.
• When ready, write new apps to target the Work Queue API directly.
Overview

• Experience with Concurrent Applications
  – Makeflow, Weaver, Work Queue

• **Thesis: Convergence of Models**
  – Declarative Language
  – Directed Graphs of Tasks and Data
  – Shared Nothing Architecture

• Open Problems
  – Transaction Granularity
  – Where to Parallelize?
  – Resource Management

• Concluding Thoughts
Scalable Computing Model

Weaver
for x in list f(g(x))

Shared-Nothing Cluster

Makeflow

Work Queue
Scalable Computing Model

Declarative Language
for x in list f(g(x))

Dependency Graph

Shared-Nothing Cluster

Independent Tasks
Convergence of Worlds

• Scientific Computing
  – Weaver, Makeflow, Work Queue, Cluster
  – Pegasus, DAGMan, Condor, Cluster
  – Swift-K, (?), Karajan, Cluster

• High Performance Computing
  – SMPSS->JDF->DAGue->NUMA Architecture
  – Swift-T, (?), Turbine, MPI Application

• Databases and Clouds
  – Pig, Map-Reduce, Hadoop, HDFS
  – JSON, Map-Reduce, MongoDB, Storage Cluster
  – LINQ, Dryad, Map-Reduce, Storage Cluster
Thoughts on the Layers

• Declarative languages.
  – Pros: Compact, expressive, easy to use.
  – Cons: Intractable to analyze in the general case.

• Directed graphs.
  – Pros: Finite structures with discrete components are easily analyzed.
  – Cons: Cannot represent dynamic applications.

• Independent tasks and data.
  – Pros: Simple submit/wait APIs, data dependencies can be exploited by layers above below.
  – Cons: In most general case, scheduling is intractable.

• Shared-nothing clusters.
  – Pros: Can support many disparate systems. Performance is readily apparent.
  – Cons: requires knowledge of dependencies.
Common Model of Compilers

• Scanner detects single tokens.
  – Finite state machine is fast and compact.
• Parser detects syntactic elements.
  – Grammar + push down automata. LL(k), LR(k)
• Abstract syntax tree for semantic analysis.
  – Type analysis and high level optimization.
• Intermediate Representation
  – Register allocation and low level optimization.
• Assembly Language
  – Generated by tree-matching algorithm.
Overview

• Experience with Concurrent Applications
  – Makeflow, Weaver, Work Queue

• Thesis: Convergence of Models
  – Declarative Language
  – Directed Graphs of Tasks and Data
  – Shared Nothing Architecture

• Open Problems
  – Transaction Granularity
  – Where to Parallelize?
  – Resource Management

• Concluding Thoughts
Observation:

Generating parallelism is easy but making it *predictable* is hard!
Challenge: Transaction Granularity

• Commit every action to disk. (Condor)
  + Makes recovery from any point possible.
  - Significant overhead on small tasks.
• Commit only completed tasks to disk. (Falkon)
  - Cannot recover tasks in progress after a failure.
  + Fast for very small tasks.
- Extreme: Commit only completed DAG.
- Problem: Choice changes with workload!
Challenge: Where to Parallelize?
Challenge: Resource Management
The Ideal Picture
What actually happens:

1 TB

128 GB

GPU

3M files of 1K each

X 1000
Some reasonable questions:

• Will this workload at all on machine X?
• How many workloads can I run simultaneously without running out of storage space?
• Did this workload actually behave as expected when run on a new machine?
• How is run X different from run Y?
• If my workload wasn’t able to run on this machine, where can I run it?
End users have no idea what resources their applications actually need.

and...

Computer systems are terrible at describing their capabilities and limits.

and...

They don’t know when to say NO.
dV/dt: Accelerating the Rate of Progress Towards Extreme Scale Collaborative Science

Miron Livny (UW), Ewa Deelman (USC/ISI), Douglas Thain (ND), Frank Wuerthwein (UCSD), Bill Allcock (ANL)

... make it easier for scientists to conduct large-scale computational tasks that use the power of computing resources they do not own to process data they did not collect with applications they did not develop ...
Categories of Applications

Concurrent Workloads

Static Workloads

Regular Graphs

Irregular Graphs

Dynamic Workloads

while (more work to do)
{
    foreach work unit {
        t = create_task();
        submit_task(t);
    }
    t = wait_for_task();
    process_result(t);
}
Data Collection and Modeling

Task Record
RAM: 50M
Disk: 1G
CPU: 4 C

Records From Many Tasks
RAM: 50M
Disk: 1G
CPU: 4 C

Task Profile
RAM
min
typ
max

Workflow Schedule
A
B
C
D
E
F

Workflow Profile

Workflow Structure
A
B
C
D
E
F
while (more work to do) {
    foreach work unit {
        t = create_task();
        submit_task(t);
    }
    t = wait_for_task();
    process_result(t);
}
Completing the Cycle

Allocate Resources

CPU: 10s
RAM: 16GB
DISK: 100GB

Measurement and Enforcement

Exception Handling
Is it an outlier?

Historical Repository

Observed Resources

CPU: 5s
RAM: 15GB
DISK: 90GB

min typ max
Complete Workload Characterization

We can approach the question:
Can it run on this particular machine?
What machines could it run on?

X 1000

128 GB
32 cores
X 1
128 GB
32 cores
X 1
X 10
16 GB
4 cores
16 GB
4 cores
X 100

12 hours
500 Gb/s I/O
1 hour
5 Tb/s I/O
At what levels of the model can resource management be done?
Overview

• Experience with Concurrent Applications
  – Makeflow, Weaver, Work Queue

• Thesis: Convergence of Models
  – Declarative Language
  – Directed Graphs of Tasks and Data
  – Shared Nothing Architecture

• Open Problems
  – Transaction Granularity
  – Where to Parallelize?
  – Resource Management

• Concluding Thoughts
Scalable Computing Model

Weaver
for x in list f(g(x))

Work Queue
Workers

Makeflow

Work Queue Master
An exciting time to work in distributed systems!
Talks by CCL Students This Weekend

• Casey Robinson, **Automated Packaging of Bioinformatics Workflows for Portability and Durability Using Makeflow**, WORKS Workshop, 4pm on Sunday.

• Patrick Donnelly, **Design of an Active Storage Cluster File System for DAG Workflows**, DISCS Workshop on Monday.
Acknowledgements

dV/dT Project PIs
- Bill Allcock (ALCF)
- Ewa Deelman (USC)
- Miron Livny (UW)
- Frank Weurthwein (UCSD)

CCL Staff
- Ben Tovar

CCL Graduate Students:
- Michael Albrecht
- Patrick Donnelly
- Dinesh Rajan
- Casey Robinson
- Peter Sempolinski
- Nick Hazekamp
- Haiyan Meng
- Peter Ivie
The Cooperative Computing Lab

University of Notre Dame

http://www.nd.edu/~ccl