Parallel and Distributed Application Paradigms

Daniel S. Katz
Senior Fellow, Computation Institute (University of Chicago & Argonne National Laboratory)
Affiliate Faculty, CCT (LSU)
Adjunct Associate Professor, ECE (LSU)
Montage (http://montage.ipac.caltech.edu/)

- An astronomical image mosaic service for the National Virtual Observatory (VAO)
- Originally developed by core team at JPL and Caltech (IPAC & CACR) – now at U. Chicago, too...; grid architecture developed in collaboration with ISI; further research with other institutions
- Delivers custom, science grade image mosaics
  - An image mosaic is a combination of many images containing individual pixel data so that they appear to be a single image from a single telescope or spacecraft
  - User specifies projection, coordinates, spatial sampling, mosaic size, image rotation
  - Preserve astrometry (to 0.1 pixels) & flux (to 0.1%)
- Modular, portable “toolbox” design
  - Loosely-coupled engines for image reprojection, background rectification, co-addition
    - Flexibility; e.g., custom background algorithm; use as a reprojection and co-registration engine
  - Each engine is an executable compiled from ANSI C
Montage use by Spitzer Legacy Teams

Color composite of co-registered 2MASS and MSX. Each square is 0.5° x 0.5°

3-color GLIMPSE image mosaic over 1.1° x 0.8°

Right: Spitzer IRAC 3 channel mosaic (3.6μm in green, 4.5μm in red, and i-band optical in blue); high redshift non-stellar objects are visible in the full resolution view (yellow box).
Central to the algorithm is accurate calculation of the area of spherical polygon intersection between two pixels (assumes great circle segments are adequate between pixel vertices).

Arbitrary Input Image

Input pixels projected on celestial sphere

Output pixels projected on celestial sphere

FITS header defines output projection

Reprojected Image
Montage Workflow

mProject 1  mProject 2  mProject 3

mDiff 1 2  mDiff 2 3

mFitplane D_{12}  mFitplane D_{23}

ax + by + c = 0  dx + ey + f = 0

mConcatFit

mBackground 1  mBackground 2  mBackground 3

mAdd 1  mAdd 2

ax + by + c = 0  dx + ey + f = 0  

mBgModel

Final Mosaic (Overlapping Tiles)

Final Mosaic (Overlapping Tiles)
Montage Initial Version

- Tested on a variety of architectures and problems
  - Good science results
- Decent performance on a single processor
- MPI version (enabled through preprocessor directives)
  - Mostly round-robin parallelization of tasks in a stage
    - Good scaling until problem/processor gets too small
  - Dynamic: each stage uses files from previous stage(s)
  - mAdd parallelization a bit more complicated
    - Scaling depends on shared I/O system performance
Many Task Computing (MTC)

- **MTC applications**: a set of tasks, usually linked by file system operations
  - Tasks are usually sequential or parallel executables
  - Set of tasks can be static or dynamic, homogeneous or heterogeneous
  - Files output by one program are used as input by others
- **MTC applications** can be described in many ways, including as in a compiled program, as a DAG, in a script, etc.
- **MTC applications** can include:
  - Bag of tasks, MapReduce, multi-stage workflow, iterative MapReduce, campaign, coupled components, ...

![MTC Application space diagram](image-url)
MTC domains

Increasing Capabilities in Computational Science

"Complexity"

1. Dimensions
2. Short
3. Time
Coarse
Long
Resolution
Fine
Coupled (and non-linear) Equations
Adaptive
Simple
Algorithms
Many
Complex
Parameters or Ensemble Members
Error Analysis
No
Optimization
No
Yes
Yes

Domain of Interest

"Time"
MTC Application Challenges

- Goal: Easily program and efficiently execute MTC applications on current and future parallel and distributed computers

- Needs:
  - Tools for describing MTC applications
  - Middleware for executing MTC applications
  - Future system characteristics to improve MTC applications


Montage on the Grid Using Pegasus

Example DAG for 10 input files

Maps an abstract workflow to an executable form

Pegasus

http://pegasus.isi.edu/

Grid Information Systems

Information about available resources, data location

Condor DAGMan

Executes the workflow

MyProxy

User’s grid credentials

Grid

Data Stage-in nodes
Montage compute nodes
Data stage-out nodes
Registration nodes

Parallel and Distributed Application Paradigms
Montage Test Problem

- 2MASS data, 6x6 degree @ m101

<table>
<thead>
<tr>
<th>Stage</th>
<th># Tasks</th>
<th># In</th>
<th># Out</th>
<th>In (MB)</th>
<th>Out (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>mProject</td>
<td>1319</td>
<td>1319</td>
<td>2638</td>
<td>2800</td>
<td>5500</td>
</tr>
<tr>
<td>mImgtbl</td>
<td>1</td>
<td>1319</td>
<td>1</td>
<td>2800</td>
<td>0.81</td>
</tr>
<tr>
<td>mDiffFit</td>
<td>3883</td>
<td>7766</td>
<td>3883</td>
<td>31000</td>
<td>3900</td>
</tr>
<tr>
<td>mConcatFit</td>
<td>1</td>
<td>3883</td>
<td>1</td>
<td>3900</td>
<td>0.32</td>
</tr>
<tr>
<td>mBackground</td>
<td>1297</td>
<td>1297</td>
<td>1297</td>
<td>5200</td>
<td>3700</td>
</tr>
</tbody>
</table>
MPI vs. Pegasus Discussion

- Both MPI and Pegasus timings ignore time to start job (queuing delay)
  - MPI: queued job is shell script; Pegasus: queued job is Condor Glide-in

- Tasks are different
  - MPI: run by stages, each is a sequential or parallel job
  - Pegasus: mDag, Pegasus, then application: tasks are clustered by Pegasus/Condor

- Unused resources
  - MPI: trailing tasks in a stage
  - Pegasus: delays of up to 5 seconds from Condor/DAGman
  - Both: sequential job bottlenecks

- Accuracy
  - I/O dominates many tasks; in a multi-user environment, none of this is very precise

- Fault tolerance
  - Pegasus supports DAG restart from previous state, MPI must rollback to previous stage

- Resources
  - Pegasus can use parallel or distributed resources, MPI requires all nodes to have shared filesystem

- Performance
  - MPI finishes in 00:25:33, Pegasus finishes in 00:28:25

- Conclusion: probably use Pegasus in production
Remaining (Montage/MTC) Challenges

- Implementation and tools are not general
  - Development - could have been simpler
    - mDAG is not a simple code
    - Could have used Pegasus DAX API, but didn’t seem any simpler
    - No way to make runtime decisions based on data
  - Deployment and Execution
    - Want to use other infrastructures, such as clouds
    - Want to make runtime decisions based on resources
      - Provide better fault tolerance than rescue DAG
    - Want to control resources (e.g., networks)
Frameworks as a solution?

- **Distributed Data Intensive Applications**
  - e.g. Particle Physics, Astronomy, Bio-Informatics

- **Programming Abstractions/Patterns/Higher-level APIs**
  - e.g. MapReduce, All-Pairs, Scatter-Gather

- **Common Runtime Support**
  - e.g. Affinity, Fault Tolerance, Patterns

- **Physical Infrastructure**
  - e.g. TeraGrid/XD, Clouds, Future Data Systems
SAGA: In a thousand words..

http://saga-project.github.io/
DAG-based Workflow Applications: Extensibility Approach

Application Development Phase

Generation & Exec. Planning Phase

Execution Phase
SAGA-Montage Proof-of-concept

- Tests run
  - toy problem: m101 tutorial (0.2° x 0.2°)
  - Useful for trying things – functionality

- digedag used for planning
  - For this problem, takes about about 0.2 s – same as Pegasus

- Runs
  - Local submission using fork
  - Local submission using ssh/SAGA
  - Local submission using Condor/SAGA
    - Local submission using 2 of above 3 and 3 of above 3
  - Queen Bee submission using ssh/SAGA
  - EC2 submission using AWS/SAGA
    - Remote submission to Queen Bee and EC2 using both ssh/SAGA and AWS/SAGA
    - Local/remote submission to local, Queen Bee, and EC2 using fork, ssh/SAGA, and AWS/SAGA

- Unstudied(yet) issues:
  - Need better understanding of application performance
  - Tradeoffs between use of MPI components or sequential components?

More MTC (Montage) Issues

- Why is an explicit DAG needed?
- User customization
  - mDAG builds a DAG for a standard workflow
  - What about a non-standard workflow?
    - Experimenting with different options
      - Build a set of plates with and without background rectification
    - Changing a module
      - mAdd uses mean for co-addition, could use median or count
  - Changing mDAG.c is not reasonable for most users
- A scripted version of Montage may be better in some cases, and may be more natural for some users
• Swift is designed to compose large parallel workflows, from serial or parallel application programs, to run fast and efficiently on a variety of platforms
  – A parallel scripting system for Grids and clusters for loosely-coupled applications - application and utility programs linked by exchanging files
  – Easy to write: simple high-level C-like functional language, allows small Swift scripts to do large-scale work
  – Easy to run: contains all services for running Grid workflow, in one Java application
    o Works on multicore workstations, HPC, Grids (interfaces to schedulers, Globus, ssh)
  – Fast: efficient, scalable and flexible execution
    o Swift/K scaling: O(1M) tasks
    o Collective data management being developed to optimize I/O
    o Swift/T in development with >10x performance improvements
• Used in neuroscience, proteomics, molecular dynamics, biochemistry, economics, statistics, and more
• http://www.ci.uchicago.edu/swift

import {...}

MosaicData header "header.hdr";
Table images_tbl "images.tbl";
Image mosaic "final/mosaic.fits";
JPEG mosaic_jpg "final/mosaic.jpg";
Image projected_images[];
Table difference_tbl "diffs.tbl";
Table fits_images_tbl "fits.tbl";
Table rectification_tbl "rectification.tbl";
Table stat_tbl "stats.tbl";
Image difference_images[];
Image rectified_images[];

Image raw_image_files[] <filesys_mapper; location = "raw_dir", suffix = ".fits">;
projected_images = mProjectBatch( raw_image_files, header );
images_tbl = mImgtbl( projected_images );
difference_tbl = mOverlaps( images_tbl );
difference_images = mDiffBatch( difference_tbl, header );
fits_images_tbl = mFitBatch( difference_images, difference_tbl );
rectification_tbl = mBgModel( images_tbl, fits_images_tbl );
rectified_images = mBgBatch( projected_images, images_tbl, rectification_tbl );
mosaic = mAdd( rectified_images OR projected_images, images_tbl, header );
mosaic_jpg = mJPEG(mosaic);
Example Wrapper Code

```c
( Image proj_imgs[] ) mProjectBatch( Image raw_imgs[], MosaicData hdr )
{
    foreach img, i in raw_imgs
    {
        Image proj_img <regexp_mapper; source = @img, match = ".*\/(.*)", transform = "proj_dir/proj_\1">;
        proj_img = mProject( img, hdr );
        proj_imgs[ i ] = proj_img;
    }
}

app ( Image proj_img ) mProject( Image raw_img, MosaicData hdr )
{
    mProject "-X" @raw_img @proj_img @hdr;
}
```
Further Swift-Montage Work

- Better understand Montage performance
- Improve Swift
  - To improve Montage performance at the runtime level
  - Using AMFORA (was: AME & AMFS)

Swift-Montage Performance

Execution time distribution of Montage test problem on 512 BG/P CPU cores

- Challenge: reduce non-execution time
AMFORA

- AMFORA: an Any-scale MTC Engine and MTC runtime File System
  - https://github.com/zhaozhang/amfora

Swift Script

```swift
foreach i in files{
  temp[i] = produce(files[i]);
  output[i] = consume(temp[i]);
}
```

Task List

```
TaskID Binary Input Output Rule
0   -b /bin/produce -i files0 -o temp0 -a files0 temp0
... N-1  -b /bin/produce -i filesN -o tempN -a filesN tempN
N   -b /bin/consume -i temp0 -o output0 -a temp0 output0
... 2N-1 -b /bin/consume -i tempN -o outputN -a tempN outputN
```

Diagram:
- Login Node
- AME Submitter
- File List
- AME Dispatcher
- AMFS Metadata Server
- AME Worker
- AMFS Storage Server
General Improvements

• Base Case: Staging
  – Automatically move input files from GPFS to local RAM disk
  – Read/write to local RAM disk
  – Automatically move output files from local RAM disk to GPFS
  – Runs in 45% of the time as MPI implementation, which reads and writes files on GPFS

• Data cache
  – Automatically move input files from local RAM disk (where produced) to local RAM disk (where needed)

• Data aware scheduling
  – Launch task where input data is (on local RAM disk)
Dataflow Patterns

- Goals:
  - Identify at runtime
  - Apply optimizations

### Dataflow Patterns

- **Gather**

- **Reduce**

- **Pipeline**

- **Scatter**

- **Multicast**

<table>
<thead>
<tr>
<th>Job</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Generation</th>
<th>Data Transfer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
File Pattern Improvements

- **Collective gather**
  - Rather than moving files one at a time from sources to a single destination, create a minimum spanning tree and collectively move the files in $\log_2(n)$ stages
  - Works well when transfer time is dominated by network latency, or files are available at once

- **Asynchronous gather**
  - Worker that will run task requests all source files
  - If available, worker fetches the file; if not, worker asks for notification
  - Allows overlapping of computation and communication
  - Works well when transfer time is dominated by network bandwidth, or files are available gradually
Montage - AME & AMFS Performance

- Those bars that are less than 1 show improvements.
- GPFS base case refers to staging input/output data from/to GPFS
  - Already 45% improvement over MPI

![Graph showing performance ratios for different techniques and applications]
Montage work by others

  - Ran Montage on virtual and physical machines, including a private cloud-like system

- Montage used as prototype application by teams involved in ASKALON, QoS-enabled GridFTP, SWIFT, SCALEA-G, VGrADS, etc.
Montage Summary

- Montage is a custom astronomical image mosaicking service that emphasizes astrometric and photometric accuracy
- Public release, available for download at the Montage website: http://montage.ipac.caltech.edu/
- MPI version of Montage:
  - Baseline performance
  - Requires a set of processors with a shared file system
- Pegasus/DAGman version of Montage:
  - Almost equivalent performance to MPI version for large problems
  - Built-in fault tolerance
  - Can use multiple sets of processors
- SAGA/digedag version of Montage:
  - Starts to address IDEAS: Interoperability, Distributed Scale-Out, Extensibility, Adaptivity, Simplicity
  - Have shown that this can be done, but haven’t gotten too far yet
- Swift version of Montage
  - Flexible scripting, no explicit DAG
  - By modifying Swift runtime, can get very good parallel performance
Application Skeletons

- Goal: Represent important applications
- With only the information needed to run the application in a distributed context
- But without requiring building the actual application, or finding data files, ...
- Compactly, e.g., few parameters for Bag of Tasks
  - Input data size (could be generalized as a distribution)
  - Output data size (could be generalized as a distribution, or a function of input data size)
  - Computing time (could be generalized as a distribution, or a function of input data size)
  - Number of tasks

Challenge

- Balance the easy of programming and usage with the performance gap between Skeleton applications and real applications.
Types of Applications that can be Represented by Skeletons

- **Bag of Tasks**
  - Set of independent tasks
  - Represents MG-RAST, TIP, science gateway, ...

- **MapReduce (distributed)**
  - Set of map tasks, then reduce tasks
  - Represents HEP histograms, genomics/metagenomics, ...

- **Iterative MapReduce**
  - Represents clustering, linear algebra

- **Campaign**
  - Similar to iterative bag of tasks, but tasks can change from one iteration to the next
  - Represents Kalman filtering

- **Multi-stage Workflow (most general for now)**
  - Represents Montage, ...

- **Concurrent tasks**
  - Represents fusion simulation, climate simulation (2nd phase)
Role and Use of Application Skeletons

1. Describe application (choose skeleton, skeleton parameters)
2. Build application (create executables & data files) and description (abstract DAG)
3. Program workflow (build Swift script, run Pegasus, etc.)

Simulate application in system
Run application in system
## Skeleton Montage vs. Real Montage

<table>
<thead>
<tr>
<th></th>
<th>mProject</th>
<th>mImgtbl</th>
<th>mOverlaps</th>
<th>mDifffit</th>
<th>mConcatFit</th>
<th>mBgModel</th>
<th>mBackground</th>
<th>mAdd</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Montage</td>
<td>282.3</td>
<td>139.7</td>
<td>10.2</td>
<td>426.7</td>
<td>60.1</td>
<td>288.0</td>
<td>107.9</td>
<td>788.8</td>
<td>2103.7</td>
</tr>
<tr>
<td>Skeleton</td>
<td>281.8</td>
<td>136.8</td>
<td>10.0</td>
<td>412.5</td>
<td>59.2</td>
<td>288.1</td>
<td>106.2</td>
<td>781.8</td>
<td>2076.4</td>
</tr>
<tr>
<td>Error</td>
<td>-0.2%</td>
<td>-2.1%</td>
<td>-0.2%</td>
<td>-3.3%</td>
<td>-1.5%</td>
<td>0.03%</td>
<td>-1.6%</td>
<td>-0.9%</td>
<td>-1.3%</td>
</tr>
</tbody>
</table>

- Similar results for BLAST, CyberShake

Skeleton Questions/TODOs

- Explore other methods for specifying skeleton parameters
- Define compute time in resource independent manner
- Add to details of tasks
  - For tasks that are internally parallel: number of internal components, internal communication requirements (to be used for mapping the task to appropriate resources)
- Turn skeletons into a ‘open source’ project – more contributors, more users
Further Work

• Goal: Use MTC paradigm to develop parallel and distributed data-intensive scientific applications to utilize a broad range of systems, with the flexibility and performance that scientific applications demand, using heritage (legacy) software

• Possible solutions: Frameworks (e.g. SAGA), MTC runtime (e.g. Swift, AMFORA)

• Application challenges
  – What are the application components?
  – How are they coupled?
  – How is functionality expressed/exposed?
  – How is coordination handled?
  – How to support layering, ordering, encapsulations of components

• Framework/runtime challenges
  – Coordinate data & computing; use task placement; and optimize I/O for best performance
  – Support range of architectures and fault-tolerance
  – Support runtime (dynamic) scheduling (including networks), including use of information systems
  – Avoid duplicating things that work – e.g., Pegasus’s planner

• System challenges
  – Tradeoff of costs & rewards: balance user & system utility (time to solution vs. system utilization)
  – Impact future systems through knowledge gained

• Tool challenges
  – Application skeletons support framework/runtime & systems research
  – What other tools are needed?
Credits

- Grid Montage: Ewa Deelman, Carl Kesselman, Gurmeet Singh, Mei-Hui Su
- DPA Theme (funded by UK e-Science Institute): Shantenu Jha, Daniel S. Katz, Manish Parashar, Omer Rana, Jon Weissman
- SAGA: SAGA Team, http://saga-project.org/
- SAGA Montage: Andre Merzky, Katerina Stamou, Shantenu Jha, Daniel S. Katz
- Swift: Swift Team, http://www.ci.uchicago.edu/swift
- Swift Montage: Jon Monette, Zhao Zhang, Daniel S. Katz, Michael Wilde
- AMFORA (AME/AMFS): Zhao Zhang, part of (DOE-funded) ExM Project, led by Michael Wilde, Matei Ripeanu, Daniel S. Katz
- Application Skeletons: Zhao Zhang, part of (DOE-funded) AIMES Project, led by Shantenu Jha, Jon Weissman, Daniel S. Katz