Computation Institute

Application Skeletons: Encapsulating MTC Application Task Computation and I/O

Daniel S. Katz and Zhao Zhang

Motivation

- Computer scientists who build tools and systems need to work on real scientific applications to prove the effectiveness of their tools and systems
  - And often vary them – change problem size, etc.
- However, accessing and building real applications can be hard (and isn't really the core of their work)
  - Some applications (source) are privately accessible
  - Some data is difficult to access
  - Some applications use legacy code and are dependent on out-of-date libraries
  - Some applications are hard to understand without domain science expertise

Target

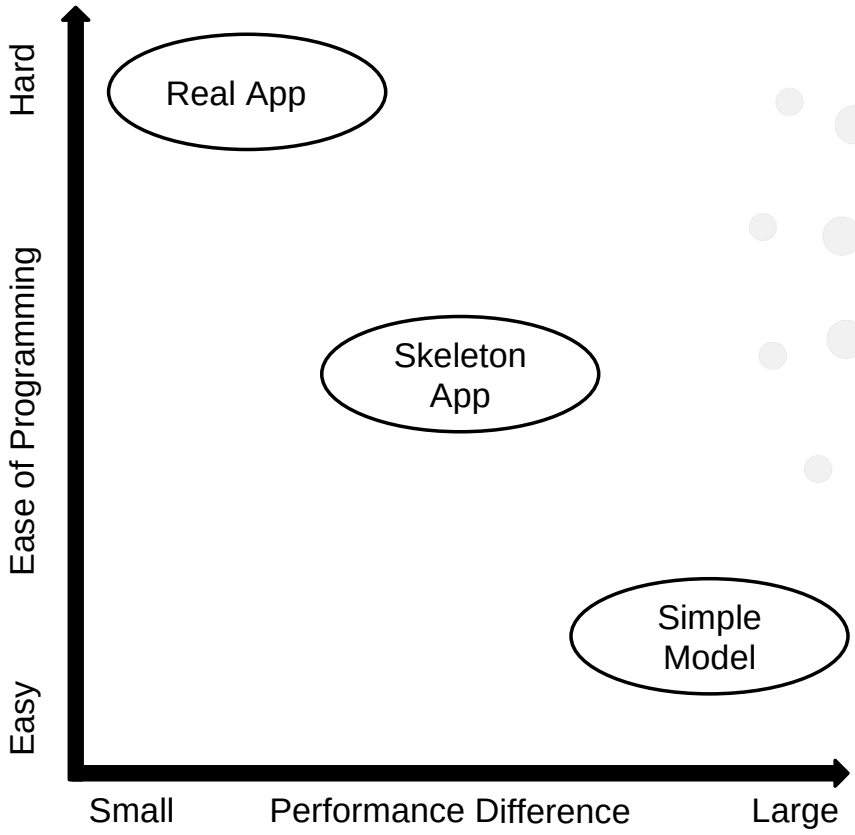
- We want to build a tool so that
  - Users can quickly and easily produce a synthetic distributed application that represents the key distributed characteristics of a real application
    - The synthetic application should have runtime, I/O, and intertask communication that are similar to those of the real application
  - The synthetic application is easy to run in a distributed environment: grids, clusters, and clouds
  - The synthetic application should be executable with common distributed computing middleware (e.g., Swift and Pegasus) as well as the ubiquitous Unix shell

Classes of Distributed Applications

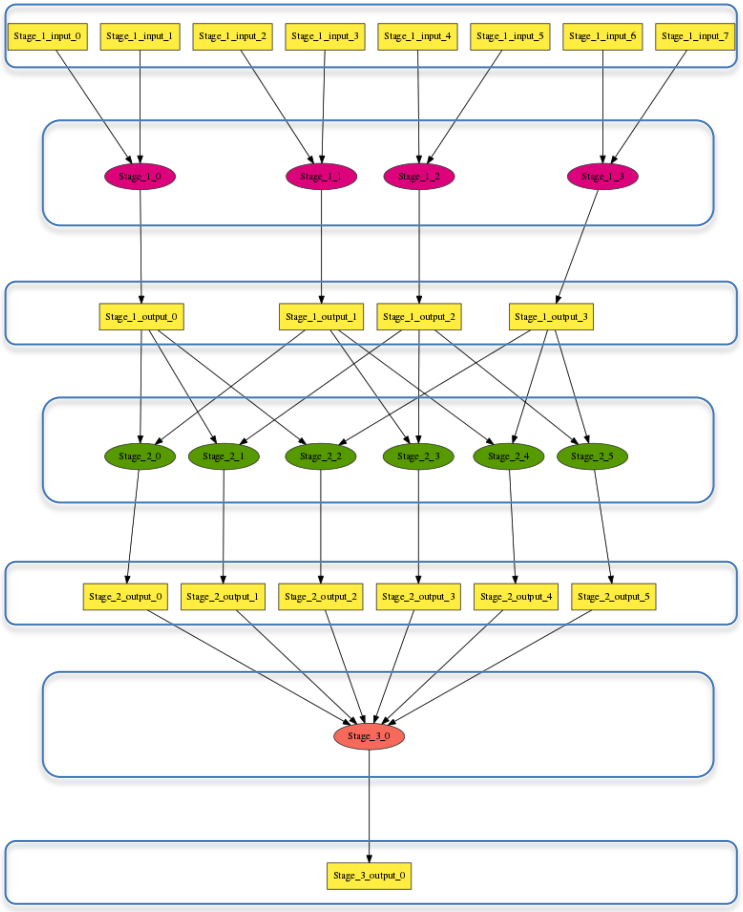
- Bag of Tasks: a set of independent tasks
- MapReduce: a set of distributed application with key-value pairs as intermediate data format
- Iterative MapReduce: MapReduce application with iteration requirement
- Campaign: an iterative application with a varying set of tasks that must be run to completion in each iteration
- **Multi-stage Workflow: a set of distributed applications with multiple stages that use POSIX files as intermediate data format**
- Concurrent Tasks: a set of tasks that have to be executed at the same time

Challenge

- Balance the easy of programming and usage with the performance gap between Skeleton applications and real applications



An Multi-Stage Application Example



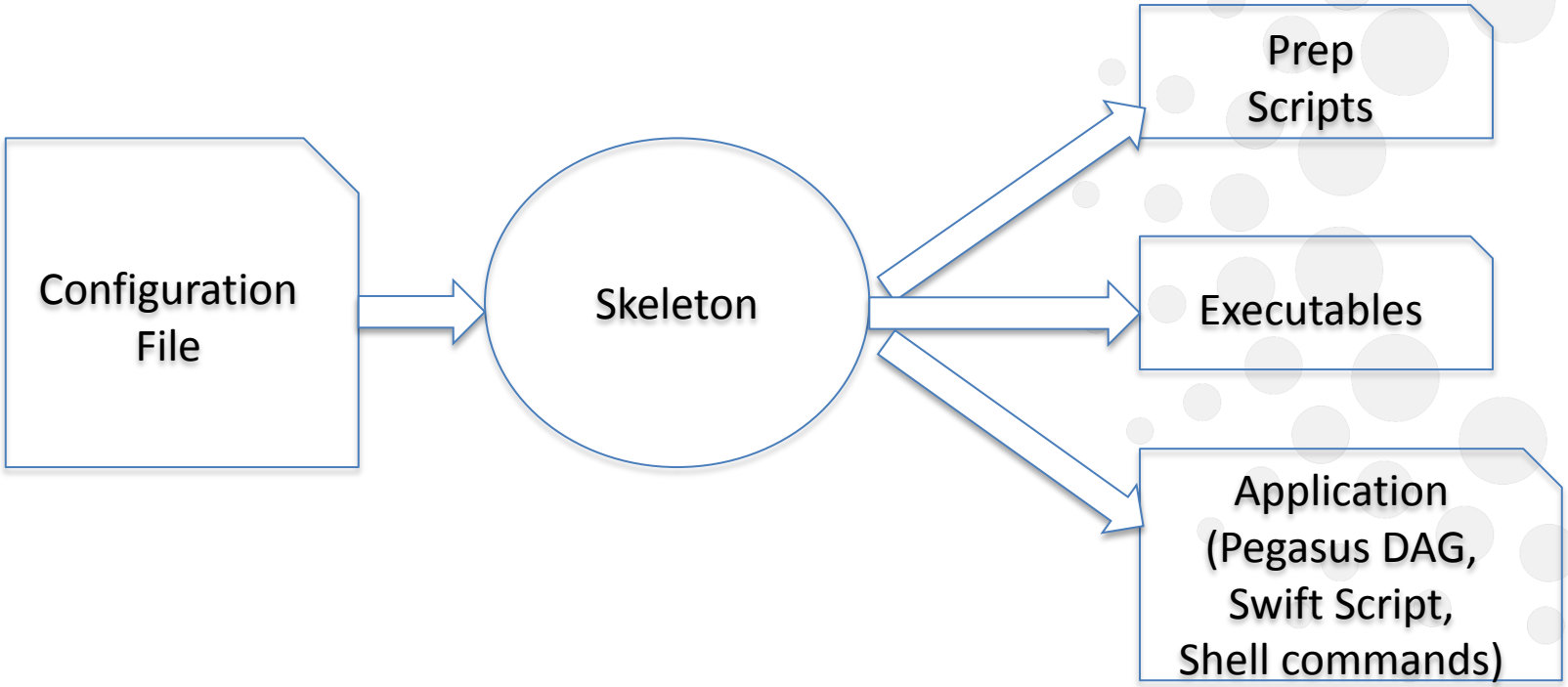
- Application have stages
- Each stage has tasks
  - Task have lengths
- Each stage has input/output files
  - Input/Output files have sizes
  - Input files map to tasks
  - Input files can be (pre) existing files or Output files from previous stages

Skeleton Abstraction

- Application Skeletons abstract an application using a top-down approach: an application is composed of stages, each of which is composed of tasks.
- An application can be defined by a configuration file containing:
  - Number of stages
  - For each stage
    - Tasks (number and length)
    - Input files (number, sizes, and mapping to tasks)
    - Output files (number, sizes)

Skeleton Tool Design

- The Skeleton tool is implemented as a parser.



Task Executable

- The current implementation of task executable copies the input files from filesystem to RAM, sleeps for some amount of time (specified as the run time), and copies the output files from RAM to filesystem
- Issues:
  - Task length described by run time does not reflect the CPU capacity
    - Different CPUs should give different performance
  - Synthetic I/O is too simple
    - Single block I/O operation may not reflect real I/O
  - Application structure is too simple
    - Performance of interleaved computation and I/O is missed

Specifying a Stage

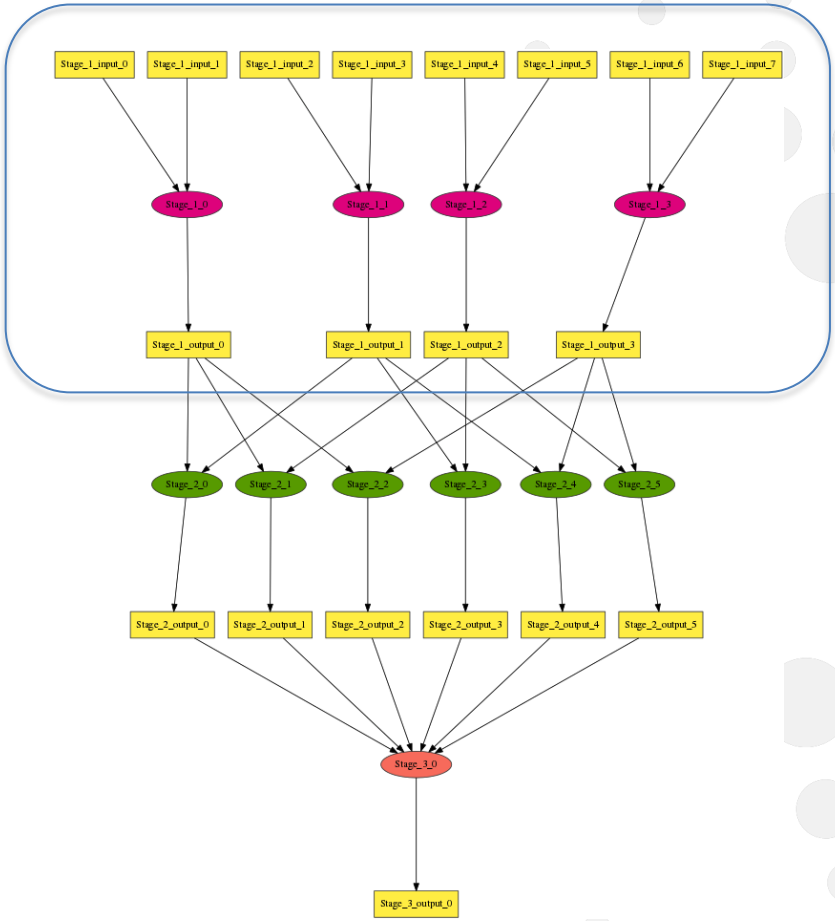
Parameter	Format	Example
Num_Tasks	Integer	16
Task_Length	dist [parameter][unit]	uniform 32s
Input_source	filesystem | Stage_$.Output	Stage_1.Output
Input_Files_Each_Task	Integer	2
Tasks_Each_Input_File	Integer	2
Input_File_Size	dist [parameter][unit]	uniform 1048576
Input_Task_Mapping	External /path/to/exec	external map.sh
Output_Files_Each_Task	Integer	2
Output_File_Size	dist [parameter][unit]	uniform 1048576

Supported distribution includes uniform, normal, triangular, and lognorm

A Multi-stage Workflow

Num_Stage = 3

Stage_Name = Stage_1
- Num_Tasks = 4
- Task_Length = normal [10, 1]s
- Input_Source = filesystem
- Input_Files_Each_Task = 2
- Tasks_Each_Input_File = 1
- Input_File_Size = normal [1048576, 1]B
- Output_Files_Each_Task = 1
- Output_File_Size = uniform 1048576B



A Multi-stage Workflow

Num_Stage = 3

Stage_Name = Stage_2

Num_Tasks = 6

Task_Length = uniform 32s

Input_Source = Stage_1.Output

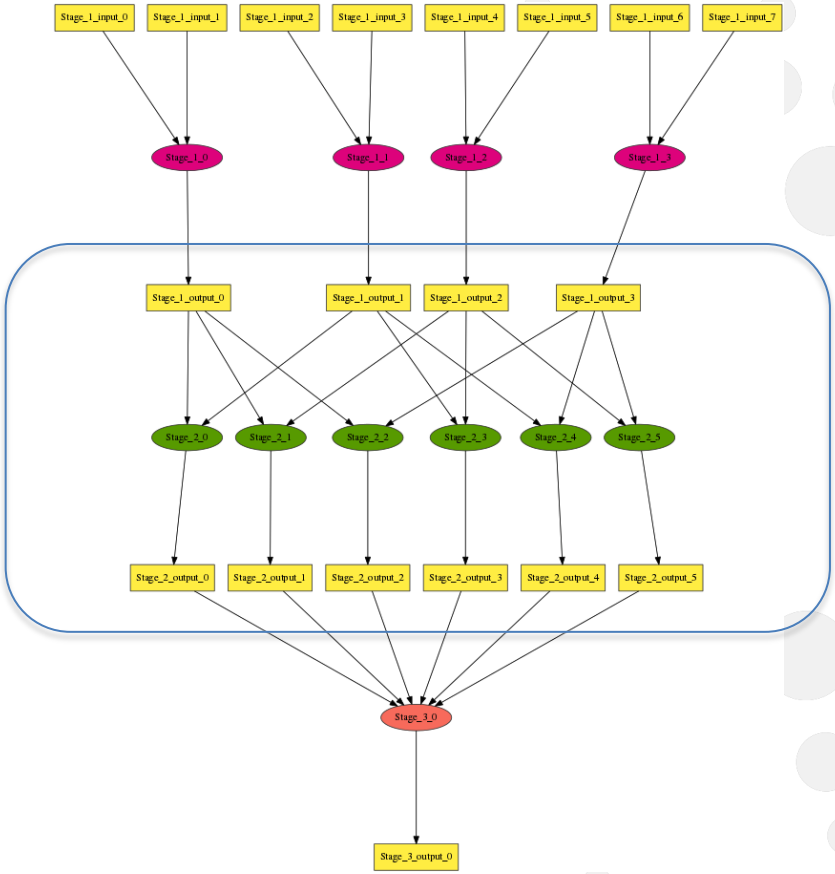
Input_Files_Each_Task = 2

Tasks_Each_Input_File = 3

Input_Task_Mapping = external mapper.sh

Output_Files_Each_Task = 1

Output_File_Size = uniform 1048576B



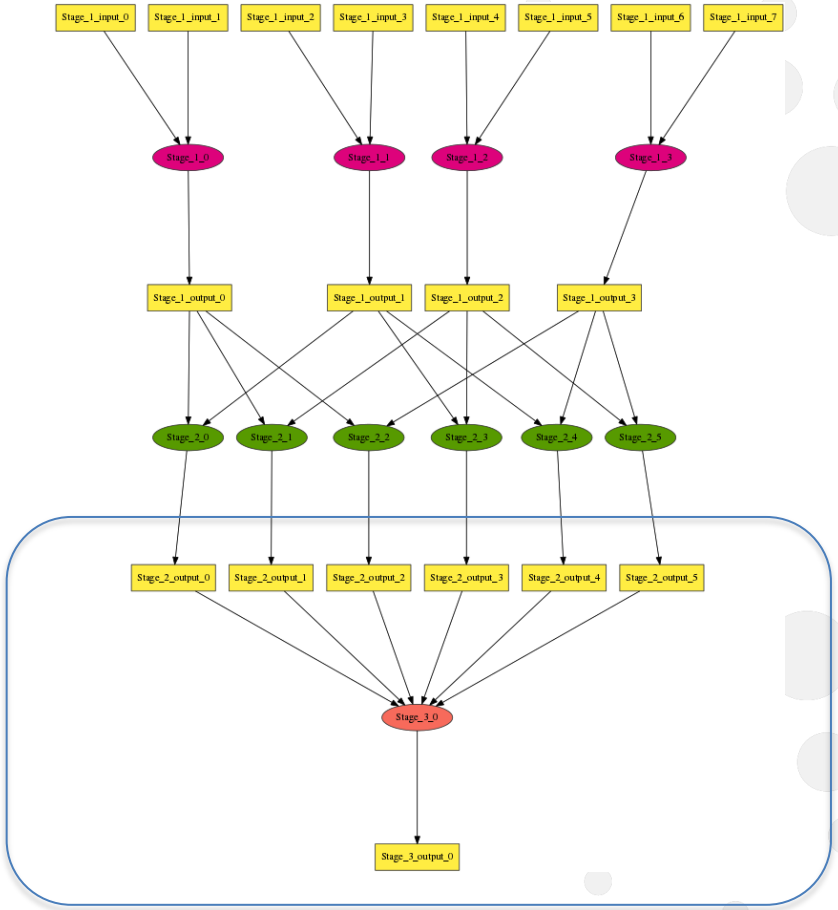
Mapping Inputs to Tasks

- If
  number of input files = N * number of tasks (N = 1, 2, ...)
  or
  number of tasks = 1
  mapping of inputs to tasks is trivial
- Otherwise, the Input_Task_Mapping option lets users specify the mapping through a Linux executable (used for the example here)
  - Example: Input_Task_Mapping = external mapper.sh
  - Output of mapper.sh
    - Stage_1_output_0 Stage_1_output_1
    - Stage_1_output_0 Stage_1_output_2
    - Stage_1_output_0 Stage_1_output_3
    - …

A Multi-stage Workflow

Num_Stage = 3

Stage_Name = Stage_3
- Num_Tasks = 1
- Task_Length = uniform 32s
- Input_Source = Stage_2.Output
- Input_Files_Each_Task = 6
- Tasks_Each_Input_File = 1
- Output_Files_Each_Task = 1
- Output_File_Size = uniform 1048576B



Skeleton Apps vs. Real Apps

- Applications:
  - Case 1: a 6x6 degree image mosaic in Montage
  - Case 2: the first 256 queries of NRxNR test in BLAST
- Platform configuration:
  - 64 compute nodes on IBM Blue Gene/P
  - Tasks are launched with AMFS
  - Each task stages input file from GPFS to RAM disk, execute the task, then copies the output files from RAM disk to GPFS

Montage Statistics

	# Tasks	# Inputs	# Outputs	In (MB)	Out (MB)	Time Avg	Time Stdev	Skeleton Task Length
mProject	1319	1319	2594	2800	10400	11.1	2.5	12
mImgtbl	1	1297	1	5200	0.8	N/A	0	16
mOverlaps	1	1	1	0.8	0.4	9	0	9
mDiffFit	3883	7766	7766	31000	487	1.7	0.6	2
mConcatFit	1	3883	1	1.1	4.3	14	0	14
mBgModel	1	2	1	4.5	0.07	283.1	0	284
mBackground	1297	1297	1297	5200	5200	0.4	0.08	1
mAdd	1	1297	1	5200	7400	N/A	0	519

Skeleton Montage Task Length

- We use average time-to-solution for mProject, mOverlaps, mDifffit, mConcatFit, mBackground
- mImgtbl and mAdd's input size exceeds single RAM disk, so we can not measure the time-to-solution with data in RAM disk
- However, we observe that these tasks' time-to-solution is proportional to the number of input files when the number of input files is small, so we project the time-to-solution with the full input data set based on the measured time-to-solution.

Skeleton Montage vs. Real Montage

	mProject	mImgtbl	mOverlaps	mDiffFit	mConcatFit	mBgModel	mBackground	mAdd	Total
Montage	290.4	139.7	10.2	359.2	64.6	283.3	102.6	793.4	2040.6
Skeleton	283.4	124.3	10.5	313.5	67.0	283.2	98.2	807.6	1987.6
Error	-2.4%	-11.1%	2.9%	-12.7%	3.9%	-0.04%	-4.3%	1.8%	-2.6%

BLAST Statistics

	# Tasks	# Inputs	# Outputs	In (MB)	Out (MB)	Time Avg	Time Stdev	Skeleton Task Lenth
formatdb	64	64	192	3800	4400	41.9	0.1	42
blastp	1024	4096	1024	70402	966	109.2	14.9	110
merge	16	1024	16	966	867	4.4	4.1	real length

Skeleton BLAST vs. Real BLAST

	formatdb	blastp	merge	Total
BLAST	82.1	1996.3	35.9	2114.3
Skeleton	76.2	1835.9	34.0	1946.1
Error	-7.2%	-8.0%	-2.9%	-8.0%

Conclusion

- The Skeleton tool can produce synthetic distributed applications that capture important distributed properties of real applications
- The Skeleton tool is simpler to use than real applications
- The Skeleton tool can generate applications that represent bag-of-tasks, MapReduce, and multi-stage workflows
- Skeleton applications can be run with mainstream workflow frameworks and systems: Shell, Pegasus, and Swift
- The execution comparison between the initial Skeleton Montage and BLAST against the real applications shows an acceptable difference of 2.6% and 8.0%
- At the stage level, the difference ranges from 0.04% to 12.7%, with six out of eleven stages within 5%

Future Work

- Near term plan:
  - Open source the Skeleton code with documentation
  - Invite users and contributors from the community
- Longer term plan:
  - User application trace data to (help) produce synthetic applications
  - Determine a way to represent the computational work in a task that when combined with a particular platform can give an accurate runtime for that task
  - Better support tasks with interleaved computation and I/O
  - Support tasks that are not generic single core tasks, such as those that internally include OpenMP or MPI
  - Support concurrent tasks that need to run at the same time
  - Investigate a better task-file mapping specification

Acknowledgements

- This work was supported in part by the U.S. Department of Energy under the ASCR award DE-SC0008617 (the AIMES project)
- It has benefited from discussions with Shantenu Jha, Andre Merzky, Matteo Turilli, Jon Weissman, and Lavanya Ramakrishnan
- Computing resources were provided by the Argonne Leadership Computing Facility
- Work by Katz was supported by the National Science Foundation while working at the Foundation. Any opinion, nding, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Thanks!

- Questions?