Computation Institute

Application Skeletons: Constructing and Using Abstract Many Task Applications in eScience

Daniel S. Katz[1], Andre Merzky[2], Zhao Zhang[3], Shantenu Jha[2]

[1]Computation Institute, University of Chicago & Argonne National Laboratory

[2]RADICAL Laboratory, Rutgers University

[3]AMPLab, University of California, Berkeley

d.katz@ieee.org, andre.merzky.net, zhaozhang@eecs.berkeley.edu, shantenu.jha@rutgers.edu

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  - Some applications use legacy code and are dependent on out-of-date libraries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
  - Real applications may be difficult to scale or modify to demonstrate system trends and characteristics

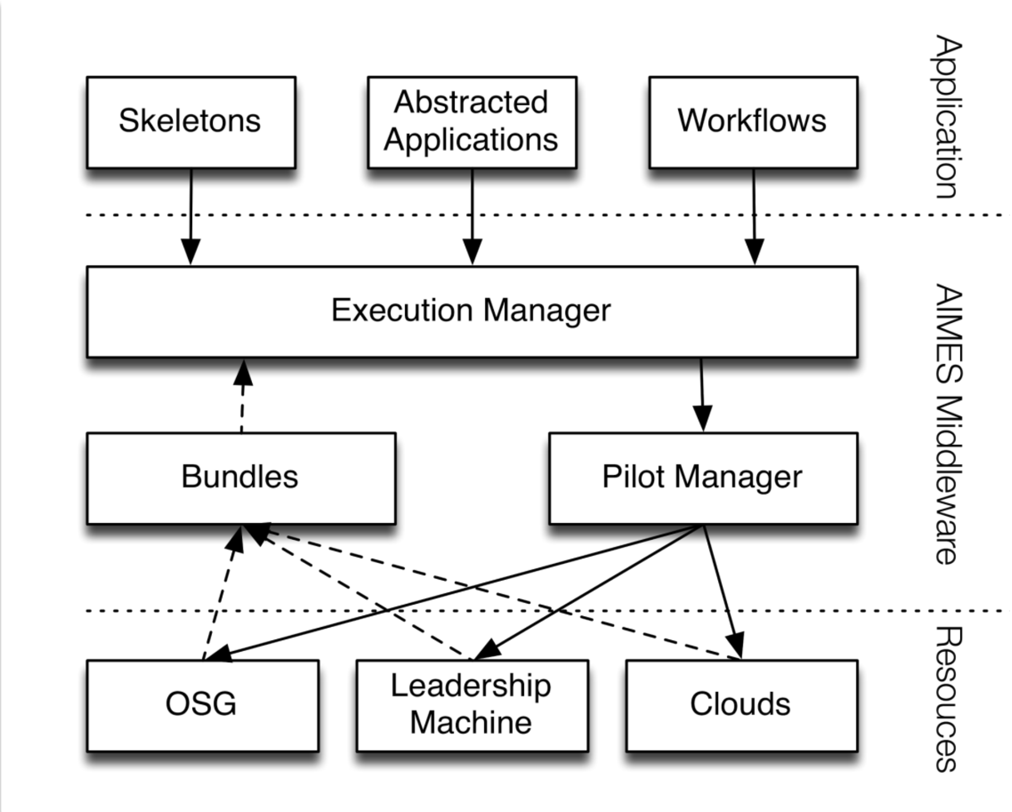
Part of the AIMES Project

AIMES: Abstractions and Integrative Middleware for Extreme Scales

- Goal
  - Improve principles and practice of distributed dynamic resource federation
- Motivation
  - Identify abstractions; implement to study & support federation
  - Improve dynamic and distributed execution on heterogeneous distributed computing infrastructure (DCI)
    - “How will my application perform on this DCI?”
    - “How can I best adapt my application to a DCI?”
    - “How can the set of resources DCI best adapt to my application?”
    - “Why did the system allocate this DCI to my application?
    - “What “variables” matter most … matter least?
  - Understand how distributed workload execution can be managed
    - Hypothesis: Using middleware that supports the integration of application-level and resource-level information
    - Questions: What are the relevant decisions, and at what level should they be made?

AIMES Approach

- Use abstractions that can be flexibly composed and support a range of experiments, including
  - **Skeletons** represent primary DCI application characteristics
  - **Resource Bundles** provide real-time info on state of diverse resources
  - **Pilot Jobs** enable dynamical distributed resource federation & management
  - **Execution strategy**: temporally ordered set of decisions that need to be made when executing a given workload



- Experiment to understand requirements & trade-offs

Skeleton 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 task type (serial or parallel), runtime, I/O buffer, I/O quantity, computation and I/O interleaving pattern 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 (not supported by current work)

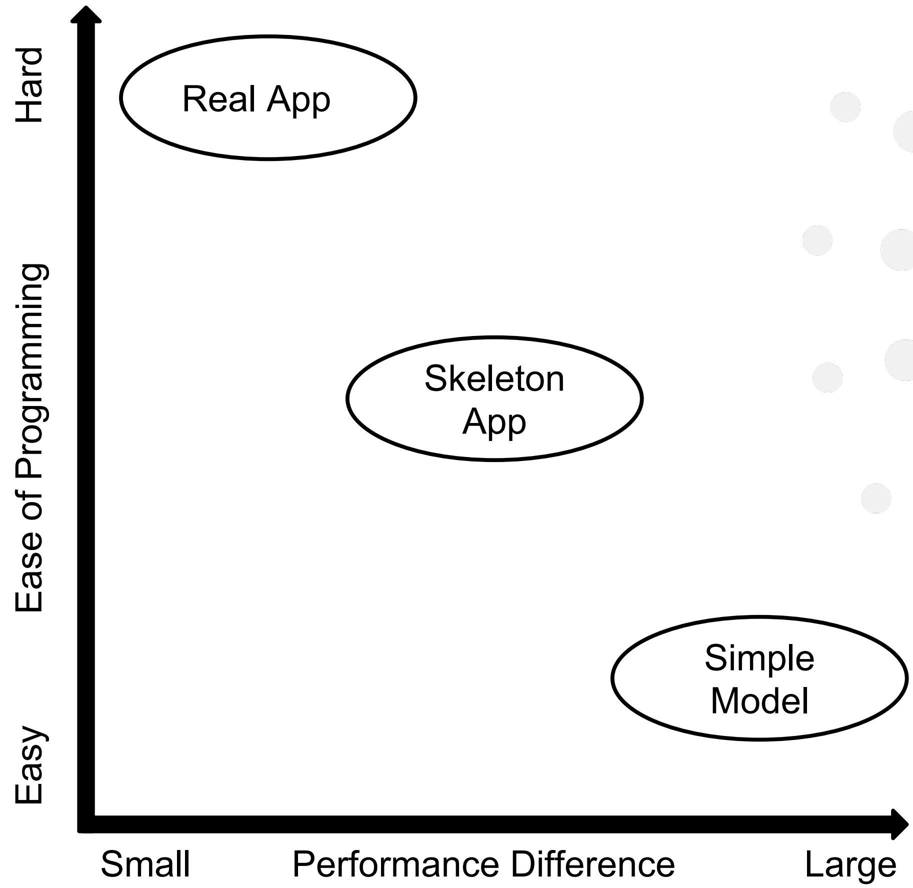
- Note that most/all of these are many-task applications

Contributions

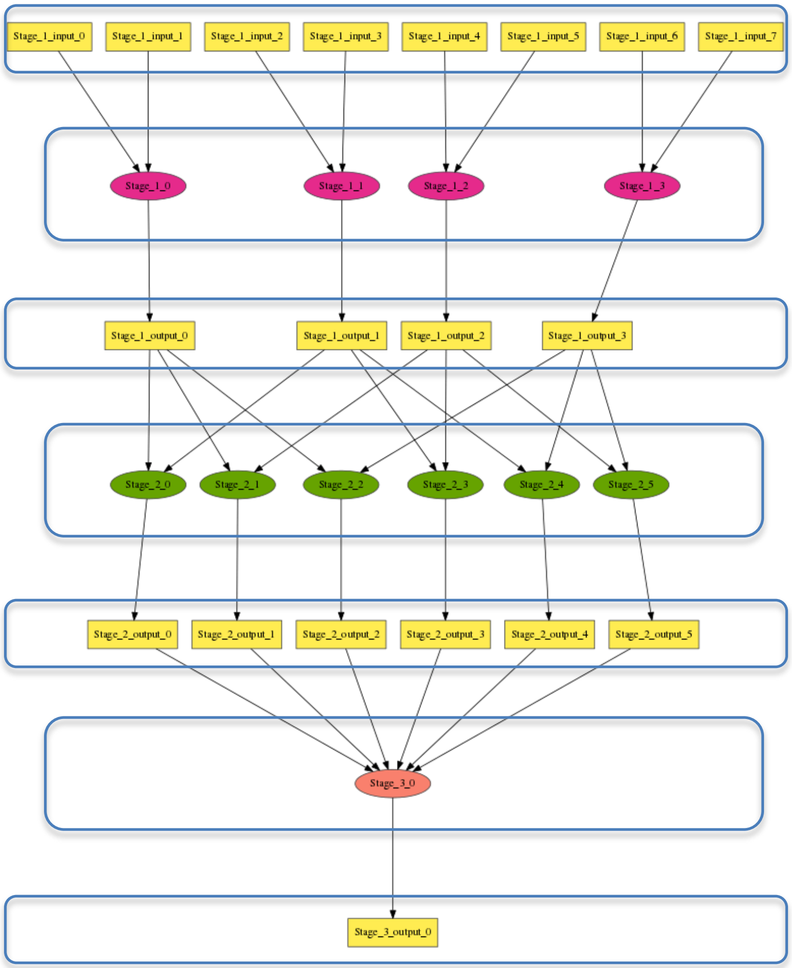
- An application abstraction that gives users good expressiveness and ease of programming to capture the key performance elements of distributed applications
- A versatile Skeleton task implementation that is configurable (number of tasks, serial or parallel tasks, amount of I/O and computation, I/O buffer size, computation and I/O interleaving options)
- An interoperable Skeleton implementation that works with mainstream workflow frameworks and systems (Swift, Pegasus, and Shell)
- The usage of Skeleton applications to simplify system optimization implementation and highlight their impacts

Challenge

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



An Multi-Stage Application Example



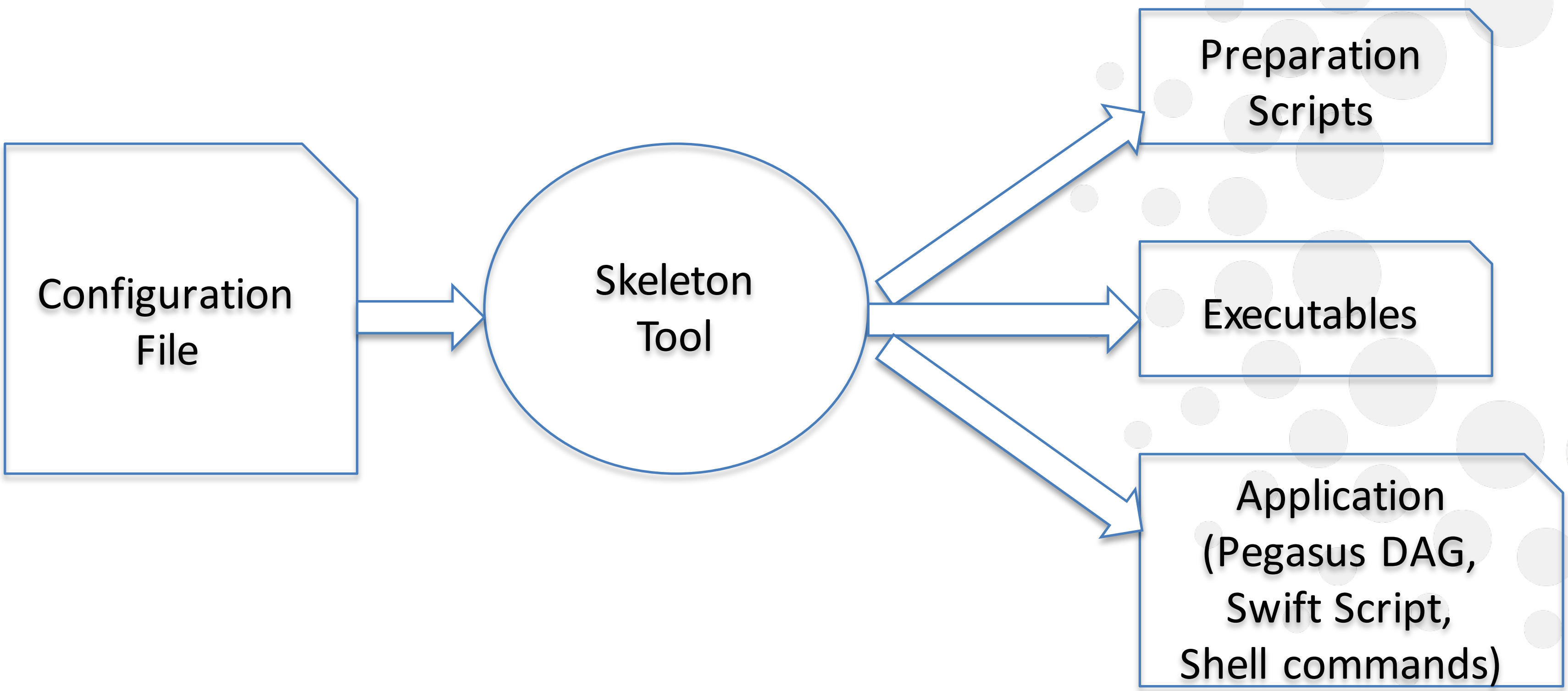
- Applications have stages
- Each stage has tasks
  - Tasks have types (serial/parallel)
  - Tasks have computation lengths
  - Input/Output files have sizes
  - I/O is through buffers
  - Input files can be (pre) existing files or Output files from previous stages
  - Computation and I/O can be interleaved
- Each stage has input/output files
  - Input files map to tasks with patterns

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
    - Task types (serial/parallel)
    - Tasks (number and computation length)
    - Number of processes for each task
    - Input files (number, sizes, and mapping to tasks)
    - Output files (number, sizes)
    - I/O buffer size
    - Computation and I/O interleaving option

Skeleton Tool Design

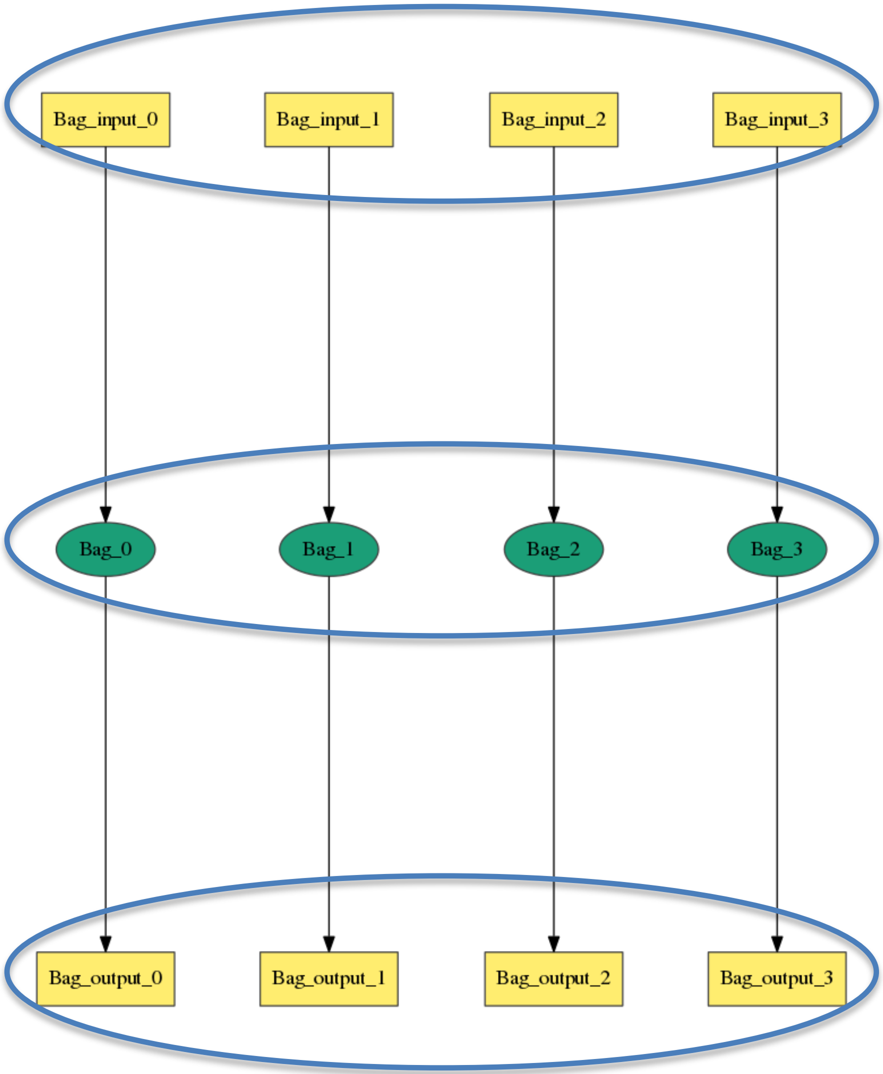
- The Skeleton tool is implemented as a parser.



Task Executable

- All tasks implemented as one standalone C program via parameters
- C program can be compiled as serial with GCC, as parallel with MPICC compiler.
- An execution example:
  - task serial 1 5 65536 65536 1 1 0 Stage_1_Input/Stage_1_Input_0_1 Stage_1_Output/Stage_1_Output_0_1 4200000
  - Path_to_Task Task_Type Num_Processes Task_Length Read_Buffer Write_Buffer Num_Input Num_Output Interleave_Option [Input_File] [Output_File Output_Size]

A Bag of Task Application Example



1. Num_Stage = 1
2.
3. Stage_Name = Bag
4. Task_Type = serial
5. Num_Processes = 1
6. Num_Tasks = 4
7. Task_Length = uniform 5
8. Read_Buffer = 65536
9. Write_Buffer = 65536
10. Input_Files_Each_Task = 1
11. Input_1.Source = filesystem
12. Input_1.Size = uniform 2100000
13. Output_Files_Each_Task = 1
14. Output_1.Size = uniform 4200000
15. Interleave_Option = 0

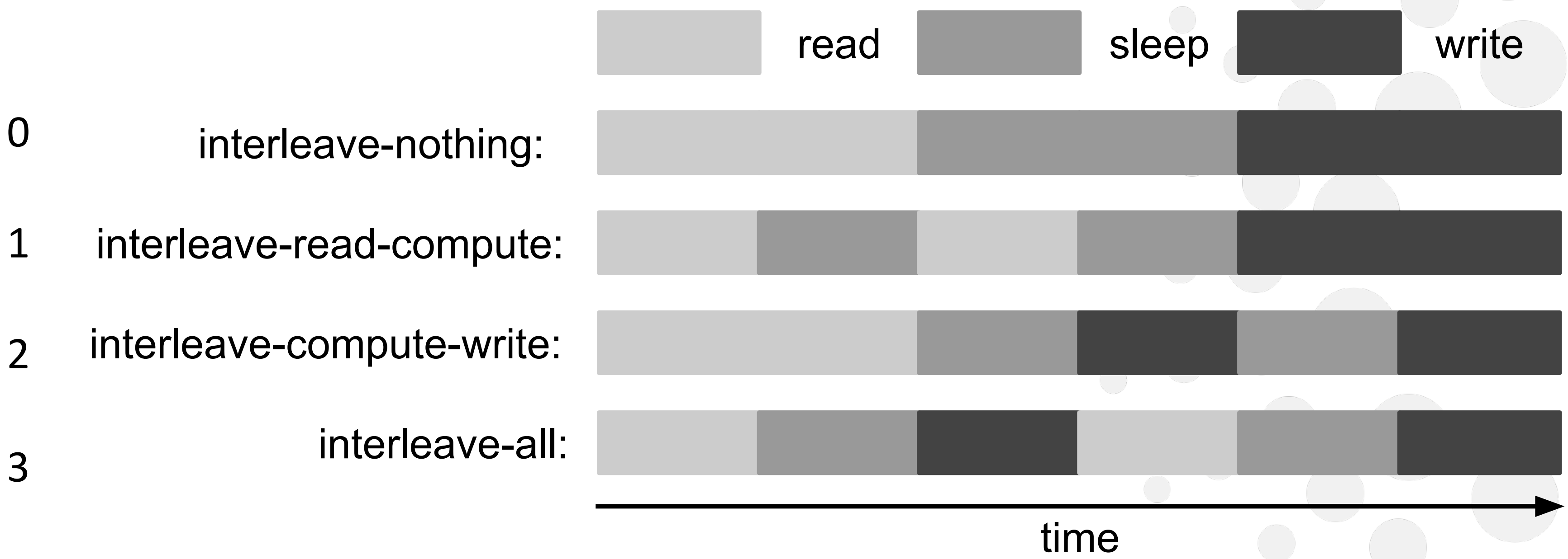
https://github.com/applicationskeleton/Skeleton/blob/master/src/sample-input/bag.input

A Bag of Task Application Example

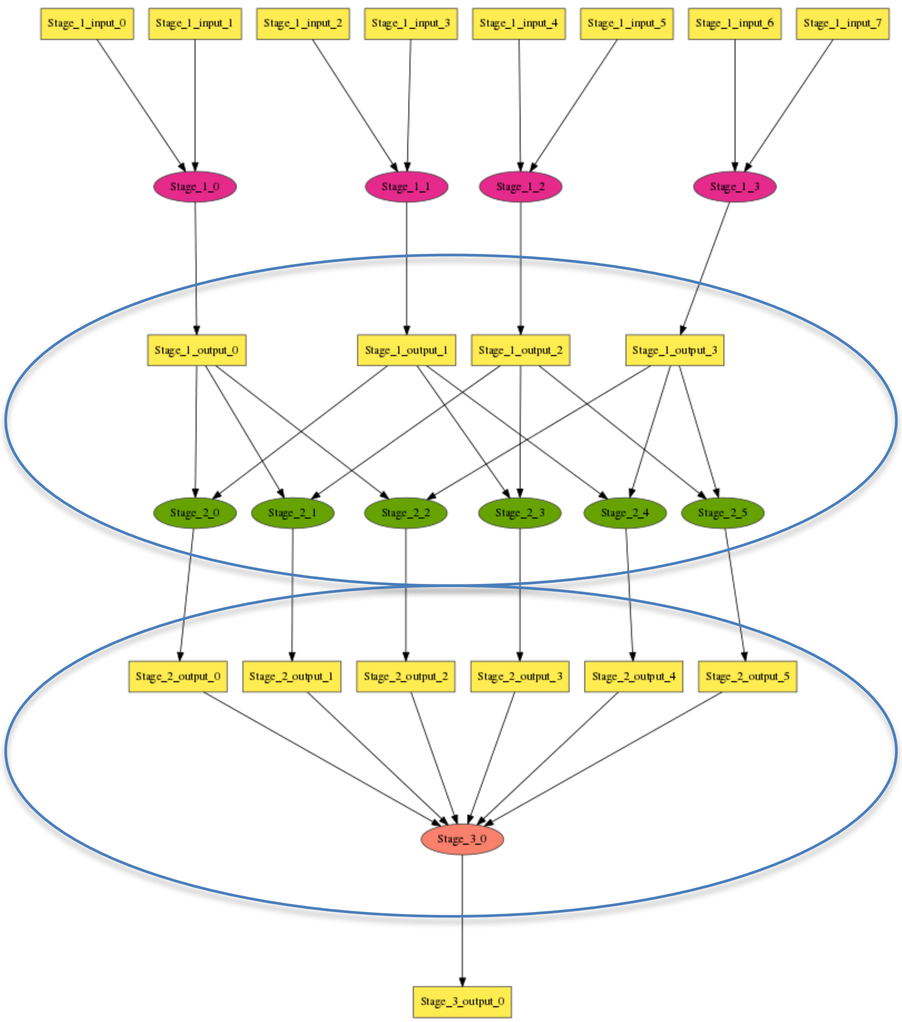
- Other options
  - Task_Type can be parallel
    - Task_Type = parallel
  - Task_Length can be a statistical distribution
    - Task_Length = normal [20, 3]
  - Task_Length can be a polynomial function of input file size
    - Task_Length = polynomial [20, 3] Input_1
    - 20*Input_1.Size^3
  - Output size can be a polynomial function of Task_Length
    - Ouptut_1.Size = polynomial [10, 2] Length
    - 10*Length^2
  - Interleaving_Option can be ...

A Bag of Task Application Example

- Interleaving_Option can be



A Multi-stage Workflow



- Num_Stage = 3
- Stage_Name = Stage_1
- …
- Stage_Name = Stage_2
- …
- Write_Buffer = 65536
- Input_Files_Each_Task = 2
- Input_Task_Mapping = combination Stage_1.Output_1 2
- …
- Stage_Name = Stage_3
- …
- Input_Files_Each_Task = 6
- Input_Task_Mapping = combination Stage_2.Output_1 6
- …

https://github.com/applicationskeleton/Skeleton/blob/master/src/sample-input/multi-stage.input

Input_Task_Mapping

- Specify Input_Task_Mapping:
  - Input_Task_Mapping = combination Stage_1_output_1 2
  - Equivalent to “N choose k” mathematically
  - 4 files, 6 tasks =>
    - file0, file1 : task0
    - file0, file2 : task1
    - file0, file3 : task2
    - file1, file2 : task3
    - file1, file3 : task4
    - file2, file3 : task5

https://github.com/applicationskeleton/Skeleton/blob/master/src/sample-input/multi-stage.input

Input_Task_Mapping

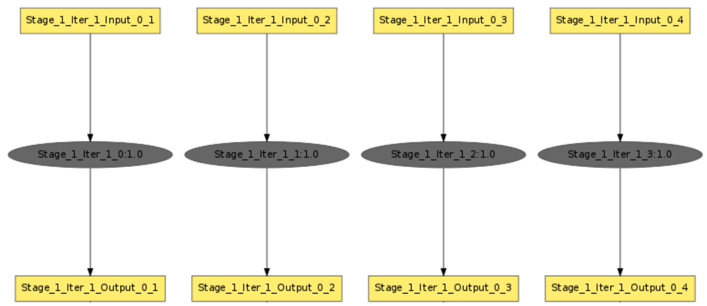
- If Input_Task_Mapping is not specified
  - Input_Files_Each_Task = 2
  - Input_1.Source = Stage_1.Output_1
  - Input_2.Source = Stage_1.Output_1
  - Files are mapped to tasks in a natural order
  - file0, file1 : task0
  - file2, file3 : task1
  - file4, file5 : task2
  - file6, file7 : task3

Input_Task_Mapping

- External mapping option
  - Input_Task_Mapping = external sample-input/mapping.sh
  - cat sample-input/mapping.sh
    - echo Stage_1_Output_0_1 Stage_1_Output_0_2
    - echo Stage_1_Output_0_1 Stage_1_Output_0_3
    - echo Stage_1_Output_0_1 Stage_1_Output_0_4
    - echo Stage_1_Output_0_2 Stage_1_Output_0_3
    - echo Stage_1_Output_0_2 Stage_1_Output_0_4
    - echo Stage_1_Output_0_3 Stage_1_Output_0_4
  - The ith line maps to the ith task

https://github.com/applicationskeleton/Skeleton/blob/master/src/sample-input/external-mapper.input

A Single Stage Iterative Application



Num_Stage = 1

Stage_Name = Stage_1
- Task_Type = serial
- Num_Tasks = 4
- Task_Length = uniform 10
- Num_Processes = 1
- Read_Buffer = 65536
- Write_Buffer = 65536
- Input_Files_Each_Task = 1
  - Input_1.Source = filesystem
  - Input_1.Size = uniform 1048576
- Output_Files_Each_Task = 1
  - Output_1.Size = uniform 1048576
- Interleave_Option = 0
- **Iteration_Num = 3**
- **Iteration_Stages = Stage_1**
- **Iteration_Substitute = Stage_1.Input_1, Stage_1.Output_1**

Stage_1.Input_1 and Stage_1.Output_1 should have IDENTICAL number of files

https://github.com/applicationskeleton/Skeleton/blob/master/src/sample-input/single-stage-iterative.input

A Multi Stage Iterative Application

```
Stage_Name = Stage_3
   Task_Type = serial
   Num_Tasks = 6
   Task_Length = uniform 32
   Num_Processes = 1
   Read_Buffer = 65536
   Write_Buffer = 65536
   Input_Files_Each_Task = 1
      Input_1.Source = Stage_2.Output_1
   Output_Files_Each_Task = 1
      Output_1.Size = uniform 1048576
   Interleave_Option = 0
```

Iteration_Num = 3

Iteration_Stages = Stage_3, Stage_4

Iteration_Substitute = Stage_3.Input_1, Stage_4.Output_1

```
Stage_Name = Stage_4
   Task_Type = serial
   Num_Tasks = 6
   Task_Length = uniform 32
   Num_Processes = 1
   Read_Buffer = 65536
   Write_Buffer = 65536
   Input_Files_Each_Task = 1
      Input_1.Source = Stage_3.Output_1
   Output_Files_Each_Task = 1
      Output_1.Size = uniform 1048576
   Interleave_Option = 0
```

Stage_3.Input_1 and Stage_4.Output_1 should have IDENTICAL number of files

https://github.com/applicationskeleton/Skeleton/blob/master/src/sample-input/multiple-stage-iterative.input

Determining Skeleton Parameters Manually

- Steps
  - Place I/O files on RAM disk, use Unix `time` command to measure run time
  - Use Unix `strace` command to find number of reads and writes, total data read and written
  - Align I/O calls in sequence order[1] to determine I/O concurrency, I/O buffer size, and interleaving option

[1]Z. Zhang, D. S. Katz, M. Wilde, J. Wozniak, I. Foster. MTC Envelope: Defining the Capability of Large Scale Computers in the Context of Parallel Scripting Applications, Proceedings of 22nd International ACM Symposium on High-Performance Parallel and Distributed Computing (HPDC'13), 2013

Determining Skeleton Parameters Automatically

- Wrap task in a profiler (uses Linux `perf`, hardware and kernel counters, system tools)
  - Measure resource consumption over time as profile: timed sequence of CPU cycles, memory and disk I/O operations
  - Dependencies between compute and I/O implicitly captured in profile in order of the sequence
- Profiles are system independent
  - Assuming comparable optimization at software and hardware levels
- Profile metrics are suitable as input for emulation

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
  - Case 3: partial seismic study of CyberShake postprocessing on site Test
- Platform configuration:
  - 64 compute nodes on IBM Blue Gene/P
  - Tasks are launched with AMFORA[1]
  - Each task stages input file from GPFS, execute the task, then writes the output files to GPFS

[1] Zhao Zhang, Daniel S. Katz, Timothy G. Armstrong, Justin M. Wozniak, and Ian T. Foster. "Parallelizing the execution of sequential scripts." Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC13). 2013.

Montage Statistics

	# Tasks	# Inputs	# Outputs	Input (MB)	Output (MB)	Skeleton Task Length (uniform)	Interleaving Option	Error in Stage
mProject	1319	1319	2594	2800	10400	11.6	0	**-0.2%**
mImgtbl	1	1297	1	5200	0.8	30.1	0	**-2.1%**
mOverlaps	1	1	1	0.8	0.4	9.1	0	**-0.2%**
mDiffFit	3883	7766	7766	31000	487	1.8	0	**-3.3%**
mConcatFit	1	3883	1	1.1	4.3	2.1	0	**-1.5%**
mBgModel	1	2	1	4.5	0.07	288	0	**0.03%**
mBackground	1297	1297	1297	5200	5200	0.4	0	**-1.6%**
mAdd	1	1297	1	5200	7400	519	0	**-0.9%**
Total								**-1.3%**

BLAST Statistics

	# Tasks	# Inputs	# Outputs	Input (MB)	Output (MB)	Skeleton Task Length	Interleaving Option	Error in Stage
split	1	1	64	3800	3800	0	3	**-1.9%**
formatdb	64	64	192	3800	4400	uniform 42	3	**-0.6%**
blastp	1024	4096	1024	70402	966	normal [109.2, 14.9]	3	**1.6%**
merge	16	1024	16	966	867	normal [4.4, 4.1]	3	**1.1%**
Total								**1.4%**

CyberShake PostProcessing Statistics

	# Tasks	# Inputs	# Outputs	Input (MB)	Output (MB)	Skeleton Task Length	Interleaving Option	Error in Stage
Extract	128	130	256	5400	11000	uniform 6.39	0	**2.6%**
Seis	4096	4352	4096	11000	96	normal [26.9, 13.3]	0	**2.4%**
PeakGM	4096	4096	4096	96	1.4	uniform 0.23	0	**2.3%**
Total								**2.4%**

Using Application Skeletons

- Data Caching
  - Comparing shared file system (PVFS) and in-memory file system (AMFORA) performance for mProjectPP
  - Using 64 n1-highmem-2 instances on Google Compute Engine (GCE)

	PVFS	AMFORA	Improvement
mProjectPP-real	285.2 seconds	100.9 seconds	**63.0%**
mProjectPP-skeleton	273.7 seconds	101.3 seconds	**64.6%**

Using Application Skeletons

- Task Scheduling
  - Data-aware scheduling vs. FIFO
  - Using 16 n1-highmem-2 GCE instances
- mProjectPP-real has 0.7% improvement
- mProjectPP-skeleton has 1.6% improvement
- Used skeletons to show: 5x larger input file size => 16.4% time-to-solution improvement with data-aware scheduling

Using Application Skeletons

- I/O Tuning
  - Multiple metadata server vs. Single metadata server
  - Using 16 n1-highmem-2 instances

- mProjectPP-real shows 1.1% improvement
- mProjectPP-skeleton shows 1.2% improvement

- Use skeletons to show: 10x shorter task length => 31.2% improvement for multiple metadata servers

Related Work

- Simplify parallel application code
  - Extract kernels: NAS Parallel Benchmarks, Berkeley Dwarfs/Motifs, CORAL benchmarks, etc.
  - Simplify non-kernel part of app: Kerbyson (SC12), Worley (SC94), miniapps (e.g., Mantevo, MADbench)
- Simplify parallel applications in time
  - Sodhi (Cluster 2008)
- Use system traces in place of applications
  - Chen (VLDB12), Harter (FAST14), Ouserhout (NSDI15)
- Skeleton-like approaches
  - Skel (for I/O), Tigres (distributed app templates), WGL (similar to our work, but simpler)
- Other work that used the term skeleton
- See FGCS paper for comparisons

Conclusion

- Skeleton tool can compose skeleton application in a top-down manner: application, stage, task
- Skeleton task abstraction allows specification of task type, task length, number of processes, I/O buffer, I/O quantity, interleaving option, and file number
- Can create easy-to-access, easy-to-build, easy-to-change, and easy-to-run bag-of-tasks, (iterative) map-reduce, and (iterative) multi-stage workflow applications
- Skeleton applications can be easily shared, making middleware and tool experiments more reproducible
- Skeleton applications have performance close to that of the real applications with an overall error of -1.3%, 1.5%, and 2.4% for Montage, BLAST, and CyberShake PostProcessing
- Skeletons can show the effectiveness of system improvements such as data caching, task scheduling, I/O tuning

Future Work

- Use application trace data to produce skeleton 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
- Support concurrent tasks that need to run at the same time to exchange information
- Test on distributed systems where latencies, particular file usage, and other issues may be more important than on the parallel systems and cloud environments

References

- Software:
  - The Skeleton tool is open source at: https://github.com/applicationskeleton/Skeleton
  - v1.2 published as: D. S. Katz, A. Merzky, M. Turilli, M. Wilde, Z. Zhang 2015. Application Skeleton v1.2., DOI: 10.5281/zenodo.13750
  - Try it! Contribute to it!
- Papers:

  - Z. Zhang and D. S. Katz, "Application Skeletons: Encapsulating MTC Application Task Computation and I/O," Proceedings of 6th Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS), 2013.
  - Z. Zhang, D. S. Katz, "Using Application Skeletons to Improve eScience Infrastructure," Proceedings of 10th IEEE International Conference on eScience, 2014. DOI: 10.1109/eScience.2014.9
  - D. S. Katz, A. Merzky, Z. Zhang, S. Jha, "Application Skeletons: Construction and Use in eScience," Future Generation Computing Systems, 2015. DOI: 10.1016/j.future.2015.10.001

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 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, finding, 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