Exploit Locality in Scientific Workflow System: A Cross-Layer Solution

Dong Dai[1], Robert Ross[2], Dounia Khaldi[3], Yonghong Yan[4], Matthieu Dorier[2], Neda Tavakoli[1], and Yong Chen[1]

[1]Department of Computer Science Texas Tech University, [2]Mathematics and Computer Science Division Argonne National Laboratory

[3]Institute for Advanced Computational Science, Stony Brook University, [4]Computer Science Department, Oakland University

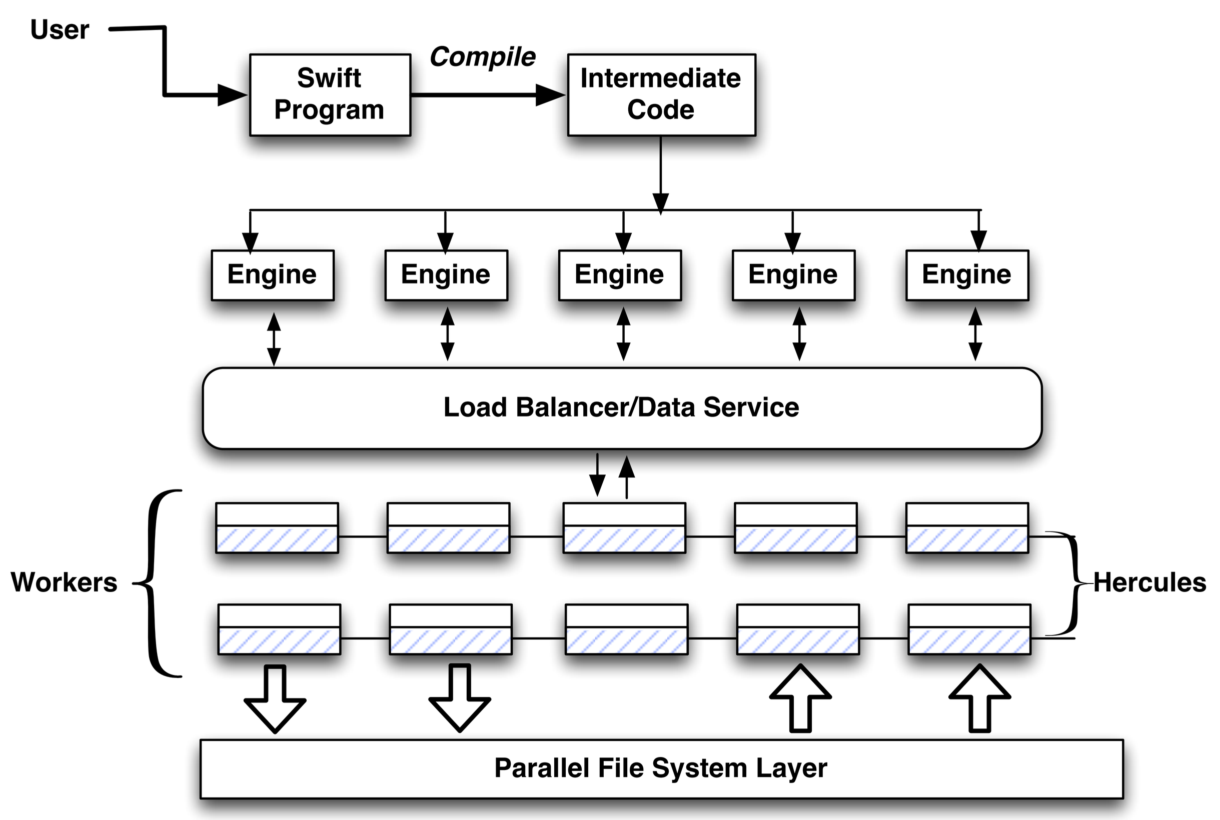
1. Motivation

- Workflow is solution for complex scientific app.
- Workflow runs upon a parallel file system (PFS)
- This leads to significant I/O bottleneck:
  - Input/Output data need to be transfered
  - Intermediate data need to be stored in PFS
  - Checkpoints need to be stored in PFS
- One solution: **Exploit the Data Locality**
  - If the data are stored in compute nodes, near the workflow tasks, we can obtain locality.
    - Build a files system in compute nodes
    - Use their local resources (memory, SSDs)

2. Background

In this research, we take Swift/T and Hercules as the basis to exploit data locality.

- Swift/T is workflow system.
- Hercules is compute-node side storage.



3. Challenges of Existing Solution: Swift/T with Hercules

```
file fs[] = glob("in/*.txt");
file outs[];

app (file out) sort (file in)
{
    "/usr/bin/sort" "-o" out in;
}

app (void signal) mpitask (file f[], string mpi)
{
    "/usr/bin/mpiexec" mpi f;
}

foreach v,i in fs
{
    file y<sprintf("out/%i",i)> = sort(v);
    outs[i] = y;
}

mpitask(outs, mpiprogram);
```

Take left program as an example, It is hard to exploit data locality in current system.

- Hercules does not support explicit control over locations. Scientists have to estimate location and tell Swift.
- Swift compiler does not collect metadata regarding data and tasks, like data size or task complexity to help schedule in runtime system.
- Swift runtime does not feedback to storage systems to direct data organization or movements.

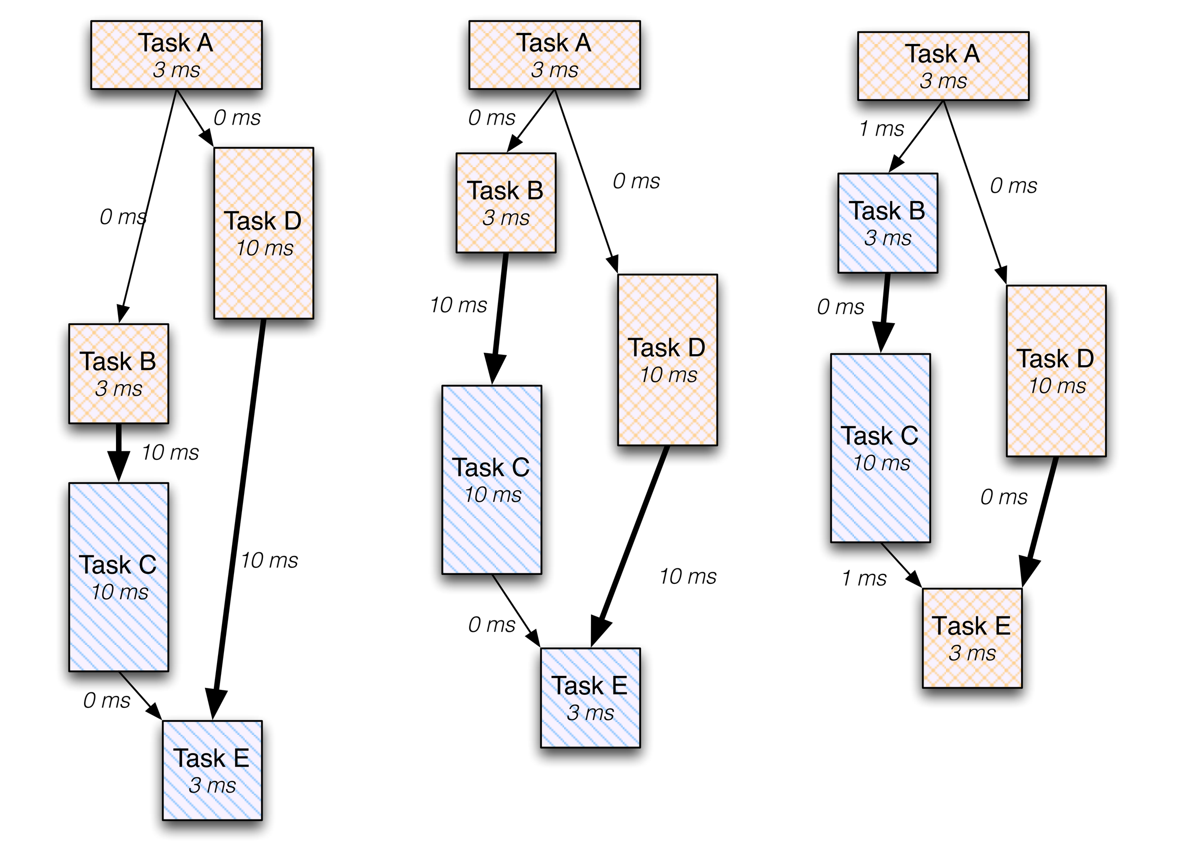
4. A Cross-Layer Solution to Exploit Data Locality

- **Location-Aware File System Extension**
  - Extend POSIX APIs to location-aware
    - ∗ Add a new mode argument (`S_LOC`) for the `O_CREAT` flag of `OPEN` function.
    - ∗ Store location information of a file in its `xattr` attributes. Obtained using `getxattr()`.
  - Build distributed location service
    - ∗ Store file in *real-loc*, but the location metadata is stored in *orig-hash-loc*
- **Hint-Assist Workflow Compiler**
    - ∗ `@size` indicates the size of an existing file.
    - ∗ `@task` indicates the key parameters for task.
    - ∗ `@compute-complex` indicates the computation cost of a task.
    - ∗ `@input-output-ratio` indicates the output size of a task regarding to its inputs.
- **Locality-Aware Workflow Scheduler**
  - A heuristic priority for task selection
    - ∗ The task that is far away from the final step in the topological order is having higher priority
  - Proactive scheduling: schedule almost-ready tasks
    - ∗ The scheduling of an almost-ready task will feedback to file system
    - ∗ File system will be pipelining intermediate results to target node

5. Preliminary Results

To demonstrate the benefit of proposed system, we conduct preliminary evaluation:

- Use a synthetic Swift workflow. Each task is manually controlled to spend certain time and generate outputs.
- Check performance under three cases: 1) default scheduler; 2) plus priority; 3) plus locality.



a) Default Scheduler b) Plus Priority c) Plus Locality

6. Summary & Future Work

- **Summary**
  - We identify the challenges of exploit data locality in scientific workflow systems even with compute-node side file system
  - We propose a cross-layer solution to tackle these challenges.
- **Future Work**
  - Full system implementation
  - Extend this cross-layer idea into task level
  - More detailed performance analysis

References

[1] Duro, F. R., Blas, J. G., Isaila, F., Carretero, J., Wozniak, J., and Ross, R. Exploiting Data Locality in Swift/T Workflows Using Hercules. In *Proc. NESUS Workshop* (2014).

[2] Wozniak, J. M., Wilde, M., and Foster, I. T. Language Features for Scalable Distributed-Memory Dataflow Computing. In *Data-Flow Execution Models for Extreme Scale Computing (DFM), 2014 Fourth Workshop on* (2014), IEEE, pp. 50–53.

Acknowledge

This material is based upon work supported by the National Science Foundation under grant CCF-1409946 and CNS-1338078.