Ubique: A New Model for Untangling Inter-task Data Dependence in Complex HPC Workflows

Jae-Seung Yeom*, Dong H. Ahn†, Ian Lumsden‡, Jakob Luettgau‡, Silvina Caino-Lores‡, Michela Taufer‡

* Lawrence Livermore National Laboratory, †NVIDIA Corporation, ‡Department of Electrical Engineering and Computer Science, University of Tennessee, Knoxville

ABSTRACT

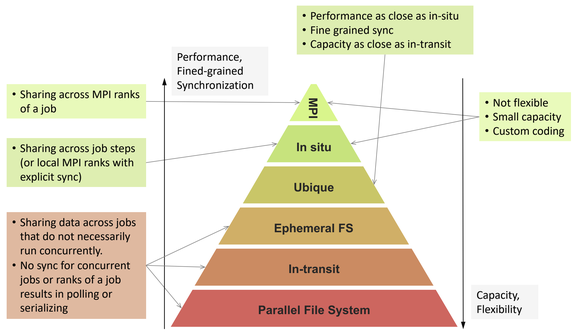
Exploiting task parallelism is getting increasingly difficult for diverse and complex scientific workflows running on High Performance Computing (HPC) systems. We argue that the difficulty rises from a void in the spectrum of existing data-transfer models for resolving inter-task data dependence within a workflow and propose a novel model, Ubique, to fill that gap.

The Ubique model combines the best from *in-transit* and *in situ* models in order for loosely coupled producer and consumer tasks to run concurrently and to resolve their data dependencies efficiently with little or no modifications to their codes, striking a balance between transparent optimization, productivity, and performance. Our preliminary evaluation suggests that Ubique can significantly outperform the parallel file system (PFS)-based model while offering transparent data transfer and synchronization which are the features lacking in many traditional models.

CHALLENGES

There exists various approaches to resolve the **inter-task data dependence**, including those based on a shared parallel file system (PFS), a ***in-transit*** model or an ***in situ*** model [1]. However, each of these approaches retains one or more of the following major drawbacks:

- **Lack of synchronization support at the file/data object level**: requires workflow themselves to synchronize consumer and producer tasks to handle cases like a consumer task attempting to read a file before the producer completes its writing.
- **Conflict with code change requirements**: Many emerging workflows compose reusable components with minimum or no code change requirement on the pre-existing programs, and hence extensive changes needed to implement the aforementioned synchronization mechanism are often a nonstarter.
- **Poor temporal/spatial locality**: Workflows use coarse grained synchronization thereby a consumer task does not start its program execution before its dependent producer finishes its entire program execution, incurring distant temporal distance to resolve a data dependency, and each file travels a long spatial distance, missing bypass opportunities.
- **Low file metadata-operation performance**: Massive numbers of small files are often employed for emerging ML-based workflows, and hence the performance of file transfers is ultimately limited by the metadata performance of the PFS.



APPROACH

Problem: Couple an application that produces data with another that consumes the data with minimum or no code change.

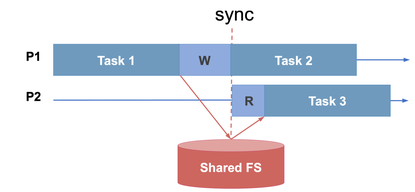
```
void producer() {
   for (i=1; I <= N; i++) {
      produce(data[i])
      write(data[i]);
   }
}
```

```
void consumer() {
   for (i=1; I <= N; i++) {
      read(data[i]);
      consume(data[i])
   }
}
```

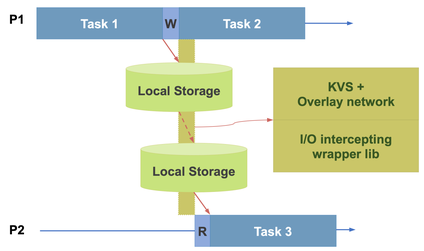
```
- description: producer task
  name: run-producer
  run:
     cmd: producer

- description: consumer task
  name: run-consumer
  run:
     cmd: consumer
     depends: [run-producer]
```

(a) Producer-consumer example (b) Maestro [2] specification in YAML

Solution 1: Traditional approach, as exemplified with Maestro, relies on a shared file system and an explicit synchronization between the end of the producer application and the start of the consumer.



Solution 2: Ubique approach relies on local storages as well as transparent data transfer between storages and synchronization per shared file.



USER INTERFACE

- Running the dyad service with FLUX [3]
  - flux exec r all flux module load dyad.so /ssd/managed_dir
- Running a user application written in C with the I/O intercepting wrapper

```
LD_PRELOAD=dyad_wrap.so:${LD_PRELOAD} ./app
```

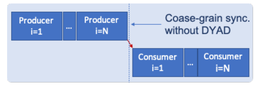
BENEFITS

No code change allows the benefits of

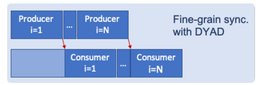
- Productivity (Fast construction of workflows with less effort)
- Easy debugging
- Portability (independent of any specific API other than widely used POSIX IO)

Performance benefits:

- Use of local storage enables faster accesses to storage, and allows avoiding metadata operation bottleneck of PFS.
- Fine-grain file level synchronization with Ubique exposes further parallelism.



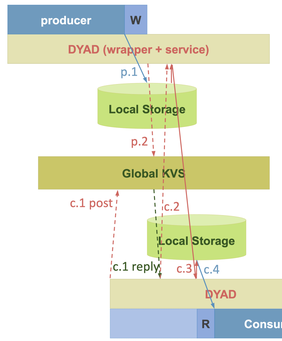
(a) Solution 1



(b) Solution 2

IMPLEMENTATION

DYAD: An embodiment of the Ubique model under FLUX [3] resource and job management system.



DYAD [4]

- DYAD module (or service) runs on each node.
- DYAD wrapper only intercepts I/O on files under the directory it manages.
  - If a file is on a local storage (LS), synchronize accesses and transfer it.
  - If it is on a shared storage, only synchronize accesses.

Producer

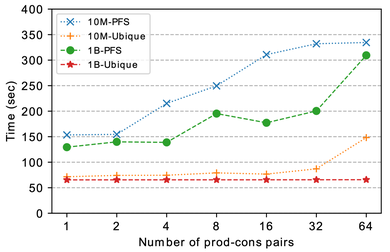
- p.1 *write(manged_dir/filepath)*
- p.2 *publish(<filepath, prod_rank>)*
  - If a file is written into ***managed_dir*** or its subdirectory, DYAD registers the filepath into the global key-value-store (KVS) of FLUX.
  - KVS entry is a pair of ***filepath*** and ***prod_rank***, where prod_rank is the FLUX rank of the service on the node where the producer is running on.

Consumer

- c.1 *query(filename)→ prod_rank*
  - Consumer queries KVS to obtain the rank of the file owner (producer). Then, blocking wait.
- c.2 *rpc_get(prod_rank, filename)*
  - Ask the owner rank to transfer the file. Once received, store it on LS
- c.3 make a copy of the data file on consumers LS
- c.4 *read(managed_dir/filename)*

RESULT

From synthetic benchmark of producer-consumer pattern



Experimental setup:

- 1 second-long computation between file I/Os.
- 1 process (either producer or consumer) per node.
- Each producer-consumer pair exchanges 64 files.
- Measured on Quartz@LLNL: Intel Xeon E5-2695 v4 w/ 128 GB
- Node local storage of DYAD: tmpfs (memory)
- Parallel File System: Lustre

CONCLUSIONS

- We propose Ubique, a new data file transfer model to enable modern HPC workflows to expose and exploit task-level parallelism.
- It transparently resolves data dependence between workflow tasks via fine-grained synchronization and direct data file transfers over network while avoiding persistent storage I/O bottlenecks, which are the features lacking in many traditional models.
- Our preliminary evaluation suggests that Ubique can significantly outperform the parallel file system (PFS)-based model.
- Ubique also offers enhanced productivity and portability of workflows.

REFERENCES

[1] C. Docan, M. Parashar, and S. Klasky, "Dataspaces: An interaction and coordination framework for coupled simulation workflows," in ACM Intl. Symp. on High Performance Distributed Computing, 2010

[2] Maestro Workflow Conductor, https://maestrowf.readthedocs.io/en/latest/

[3] D. H. Ahn, N. Bass et al., "Flux: Overcoming scheduling challenges for exascale workflows," Future Generation Computer Systems, vol. 110, pp. 202–213, 2020.

[4] DYAD repository: https://github.com/flux-framework/dyad

ACKNOWLEDGMENTS

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DEAC52-07NA27344.

LLNL-POST-840859

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52 07NA27344.