Generic Support for Bulk Operations in Grid Applications

Stephan Hirmer, Hartmut Kaiser,
Andre Merzky, *Andrei Hutanu*,
Gabrielle Allen

Center for Computation & Technology

AT LOUISIANA STATE UNIVERSITY

Outline

- Introduction
- Grid API's
- SAGA. Asynchronous operations
- Bulk operations within the SAGA C++ reference implementation
- Benchmarks. Conclusion

CENTER FOR COMPUTATION & TECHNOLOGY AT LOUISIANA STATE UNIVERSITY

Introduction

- Latencies associated with invocation of remote operations and inter-process communication affects performance
- One way to deal with this : cluster related operations in a single operation : *bulk operation*
- Issue is that some component between the user and the middleware needs to do this optimization : usually the user

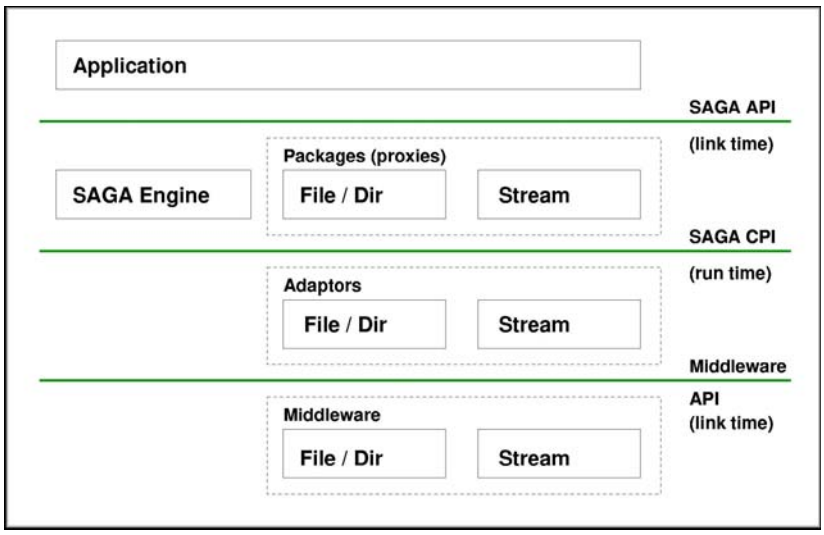
Grid APIs

- Naturally concerned with performance problems
- They usually offer means to hide the latency such as asynchronous operations (tasks) or bulk operations
- SAGA : OGF application-oriented standard
  - 80:20 rule. 80% functionality with 20% effort (complexity)

SAGA

- Covering : file access, replica management, job submission and control, and data streaming.
- API needs to be simple : optimizations are not exposed to the user
- However some use cases require these optimizations : need to show that they can be integrated while keeping the simplicity of the API

SAGA Architecture



Requirements for bulks

- Need two types of information
  - Information about task dependencies: what tasks are independent and can be run as a bulk
  - Information about the tasks themselves: which tasks are similar enough so that it makes sense for them to run together

Asynchronous operations in SAGA

- Tasks
  - Handles to asynchronous function calls
  - run(), wait(…), cancel(), get_state()
- Task Container
  - Concept to handle a group of async. function calls.
  - add_task(…), remove_task(…), run(), wait(…), …
- Task Bulks
  - A set of arbitrary tasks, sharing common properties.

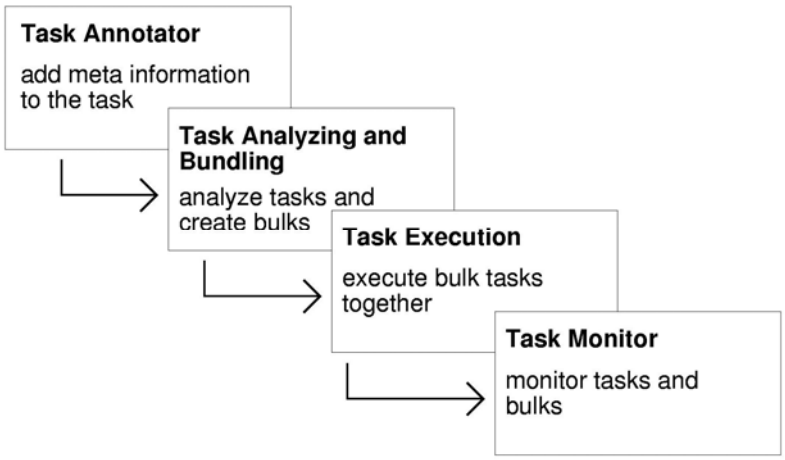
Example code

```
vector<string> files = …;
saga::task_container tc;

//create file copy tasks
while (files.size()) {
  saga::file f (files.pop());
  tc.add(f.copy<saga::task>
          (“/data/”));
}

//run all tasks
tc.run();
//wait for all tasks
tc.wait();
```


Requirements, refined

- Explicit asynchronous API
  - Synchronous operations are not considered
- Information about task (non)dependencies
  - Implicitly provided by the container class
- Information about task similarities
  - No requirements on the API but the implementation should allow inspection of the remote operation

Architecture of our system



Adding meta-information

- Not just a function pointer : Need to have access to information about the executed method (function name, parameter values, class name and instances)
- All this stored in the task and used as a basis for clustering heuristics

Task Analyzing & bundling

- task_container::run() used as entry point:
  - using meta-information for analysis
  - bundling “similar” tasks together
- according to different clustering strategies:

operation	class	bulk
same	same	yes
same	different	no
different	same	yes
different	different	no

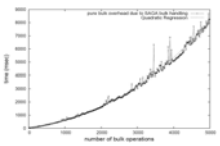
Task execution

- Using a standard selection tool an adaptor is selected. The adaptor tries to execute all the tasks using its specialized bulk handling
- returns a subset (may be empty) of tasks he couldn’t execute
- new bulk-adaptors are selected until all bulks are executed
- if necessary, fall back to one-by-one execution.

Prototype implementation

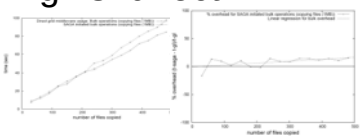
- SAGA engine was extended to allow harvesting of semantic information for operations
- Important measure : overhead of bundling and analyzing the tasks
- Important to note : this is for optimizing the invocation of the operations, not the operations themselves
- Example adaptor : interfaces to a GridFTP-based file copy (GSI) service

Benchmarks

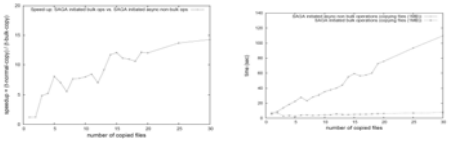


– Introduced sorting overhead

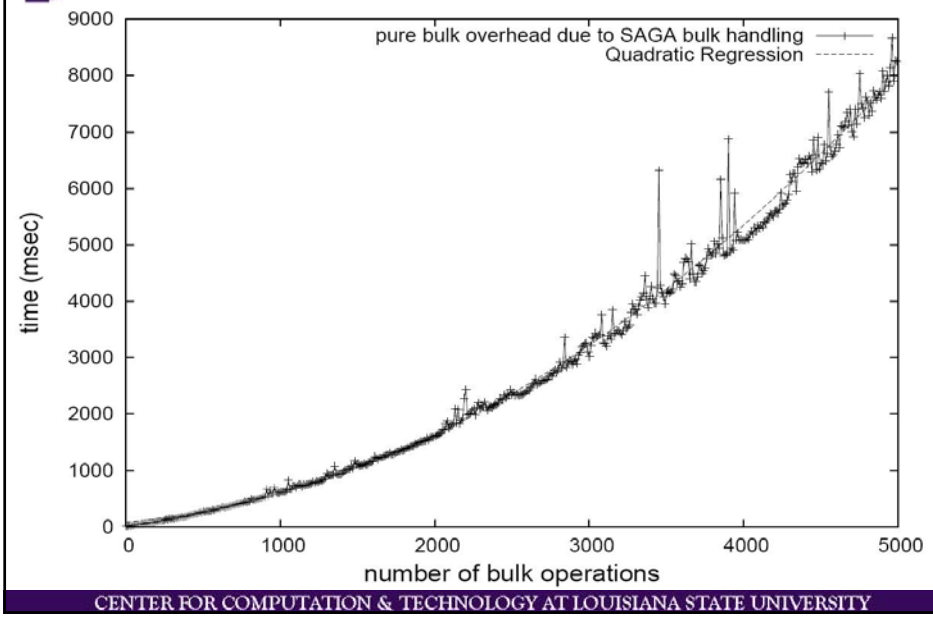
– SAGA initiated bulk handling vs. direct middleware invocation.



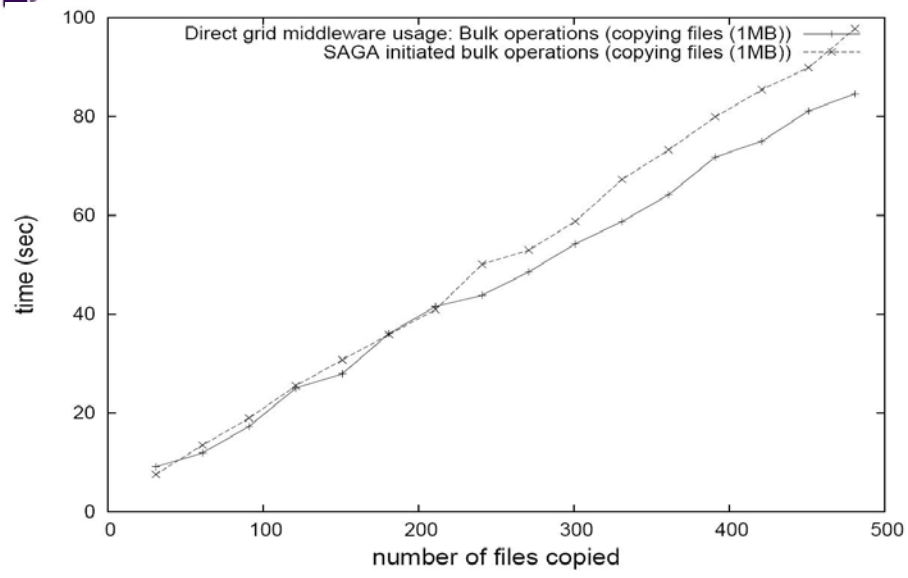
– SAGA initiated bulk handling vs. SAGA initiated async. function calls.



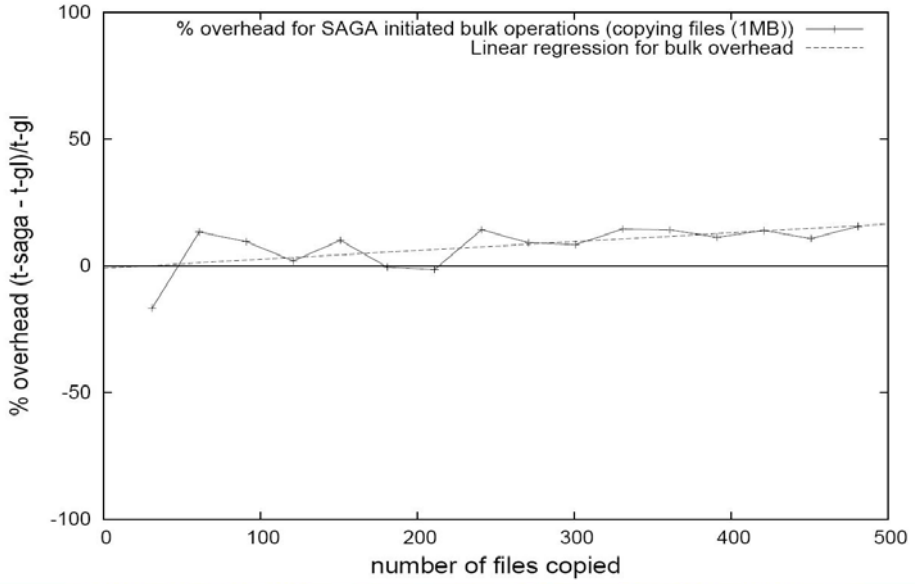
Benchmarks



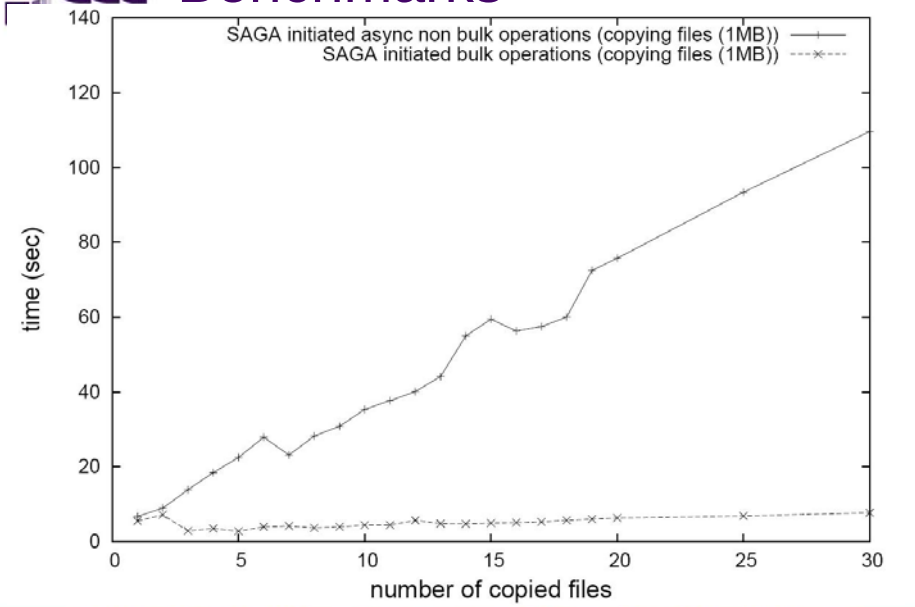
Benchmarks



Benchmarks



Benchmarks



Conclusion

- Bulk optimizations could be done within SAGA
- Three requirements for generic bulk optimizations in API implementations:
  - Asynchronous API
  - Explicit information about task dependencies
  - API implementation must be able to inspect the tasks in order to find similar tasks
- Benchmarks:
  - Minor overhead introduced, but not neglectable