Adaptive Execution 101

... for Applications in Distributed Computing Environments

Part 1: Building Blocks & Tools

Ole Weidner | ole.weidner@me.com | 11/23/11

Adaptive Execution

- **Context:** Distributed Comp. & Applications
- Goal driven, conditional *executable <-> resource mapping* and *application decomposition* based on (changing) application and resource properties
- Goals can be manifold: maximize throughput, optimize resource utilization, time-to-completion, resource or cost constraints, ...
- Sometimes referred to as *dynamic execution*, but that term is even more ambiguous

Application vs Executable

- Both terms are often used interchangeably...
- An *application* is software designed to perform a specific task. It consists of one or more *executables* or *executable instances* and possibly other things like files, databases, etc. That's why applications are usually *decomposable*!
- E*xecutables* realize a specific computational functionality of an application. They must be *mapped* to a computing resource

A Classification Attempt 1/3

- Based on *where* adaptive occurs:
  - A *centralized* adaptive execution 'component' controls all aspects of the optimization process, e.g., Condor's "match maker"
  - A set of *decentralized*, components contribute individually to the optimization process, e.g., based on agent-networks
  - Adaptive execution in *user-space* v.s. in *system-space* (or both?)

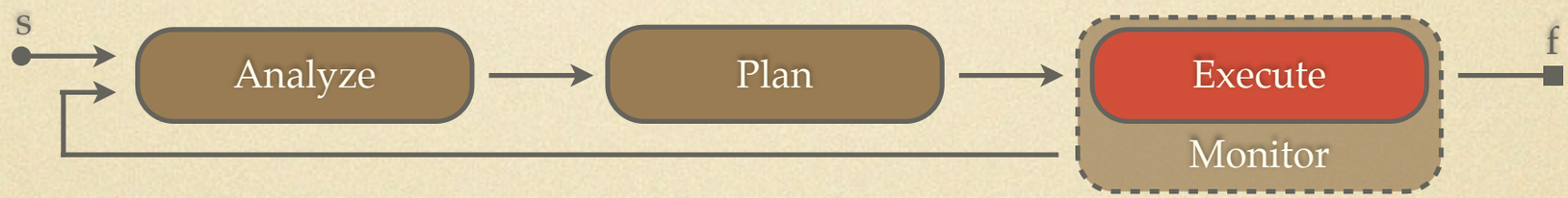
A Classification Attempt 2/3

- Based on *how* adaptivity is implemented:
  - Manually
  - Static Rules
  - Dynamic
    - Knowledge-based
    - Learning / AI based

A Classification Attempt 3/3

- Based on *when* adaptivity occurs:
  - A: *pre-execution* (one-time adaption):

  - B: *in-execution* (continuous adaption):



  - Examples: Condor (A), Pegasus (A), Adaptive-Pegasus (B)

Analyze, Plan, Execute

- *APE*: Building blocks for any class of adaptive execution system!
  - Analyze resource environment and application properties and requirements
  - Plan application decomposition, execution trajectory, resource mapping, ...
  - Execute the application according to plan ...

APE with SAGA?

- *E* - is pretty obvious - that's what the *SAGA Job API* is all about!
- **A** - not so much. *SAGA Service Discovery* (SD) and Information Service (IS) APIs might help to analyze some resource aspects. Application analysis? No.
- **P** - nope. No such thing in SAGA. Write your own planner application, tool, framework, ...

SAGA SD API

- ***saga.sd.Discoverer:*** knows and provides information about one or more services
- ***saga.sd.ServiceDescription:*** describes a service (name, type, access url, ...)
- ***saga.sd.ServiceData:*** exposes service properties. Dependent on type and backend, but Bliss tries to enforce the GLUE schema
- *http://oweidner.github.com/bliss/apidoc/*

SAGA SD Example

```
sdd = saga.sd.Discoverer("pbs+ssh://india.futuregrid.org")
services = sdd.list_services()

for service in services:
    # for each service, get some key metrics via the
    # service data object
    data = service.get_data()
    print "  * Serivce: '%s', type: '%s', url: '%s'" \
      % (service.name, service.type, service.url)
    print "     |- Running Jobs          : %s" \
      % (data.get_attribute("GlueCEStateRunningJobs"))
    print "     |- Waiting Jobs          : %s" \
      % (data.get_attribute("GlueCEStateWaitingJobs"))
    print "     |- Total CPUs            : %s" \
      % (data.get_attribute("GlueSubClusterPhysicalCPUs"))
    print "     |- Free CPUs             : %s" \
      % (data.get_attribute("GlueCEStateFreeCPUs"))
    print "     '- CPUs per Node         : %s" \
      %
(data.get_attribute("GlueHostArchitectureSMPSize"))
```

SAGA Job API

- ***saga.job.Description:*** describes the properties and requirements of an executable (job)
- ***saga.job.Service:*** represents a (remote) entity that can run jobs (e.g., a cluster)
- ***saga.job.Job:*** represent the executable itself and provides handles to control its execution (e.g., start, stop, ...)
- *http://oweidner.github.com/bliss/apidoc/*

A Simple Example 1/4

- Goals: Start application as soon as possible and use selected (HPC) resources as efficient as possible
- Resources: XSEDE, LONI and FutureGrid machines
- Application: DNA Short-read mapping against a reference genome (yes. *BFast*, of course)

A Simple Example 2/4

- **Analyze**:
  - Find out number of BFAST executable instances (defined by # read files)
  - Find out which resource has how many nodes available (saga.sd)
  - Find out the properties of the nodes, like memory, cpus, cores (saga.sd)
  - *https://github.com/oweidner/bliss/blob/master/examples/sd-api/*

A Simple Example 3/4

- **Plan**: e.g., apply a simple, static rule
  - *pick resource with the highest # of immediately available cores (nodes*cores).*
  - *If that's < # executable instances, add the resource with the 2nd highest # of immediately available cores ... and so on.*
  - *Map # cores BFast executable instances to each node*

A Simple Example 4/4

- **Execute**:
  - Create jobs for executable instances and run on the selected resources (saga.job) according to our plan

Sorry, not complete yet!

- *https://github.com/oweidner/bliss/blob/master/examples/advanced/dynamic_execution_01.py*
- **DONE**! We have just automated something using concepts of adaptive execution that many people still do by hand!

What's Next ?

- Part 2: *M* as in monitoring
- Part 3: *K* as in Knowledge
- Part 4: *MAPE-K* Framework Design
- Part 5: System Survey
- Part 6: Experimental Results

Resources

- Contact: ole.weidner@me.com
- Bliss Discussion List: http://groups.google.com/group/saga-bliss
- Bliss Wiki & Documentation: https://github.com/oweidner/bliss/wiki
- These Slides: *http://oweidner.github.com/bliss/docs/bliss_adex_101.1*