Integration of Compute-Intensive Tasks into Scientific Workflows in BeesyCluster[⋆,⋆⋆]

Paweł Czarnul

Faculty of Electronics, Telecommunications and Informatics
Gdansk University of Technology, Poland
pczarnul@eti.pg.gda.pl

Abstract. The paper presents design, implementation details and simulations of scientific workflows involving compute-intensive tasks on clusters and PCs. The author has incorporated support for scientific workflows into previously developed J2EE-based BeesyCluster, deployed at Academic Computer Center Gdansk Poland on large HPC resources including a large 288-processor Itanium2 cluster. BeesyCluster allows users to manage various accounts on clusters/PCs via WWW/Web Services, run shell interactively, compile, queue, run tasks, publish services for other users, work in teams. A frequent scenario in HPC computing is analyzed, in which a workflow is combined from tasks offered by different users. Steps of the workflow include data preparation and following simulations run in parallel on clusters, with and without queuing systems.

1 Introduction and Related Work

In today's distributed networks, services are combined into scientific workflows enabling convenient execution of frequent scenarios in various disciplines like molecular biology, bioinformatics, ecology, chemistry, often using HPC resources.

Examples of systems supporting workflows include Kepler ([1]), Pegasus ([2]), Triana ([3]), P-GRADE ([4]). Such systems may utilize Web Services or grids e.g. Kepler ([1]) features Web Service or grid actors – WEBSERVICE and GLOBUSJOB for calling Web Services and running Globus jobs respectively. Directed Acyclic Graph Manager (DAGMan) is a meta-scheduler for Condor jobs (described by a DAG).

Potential obstacles in integration in current grid ([5]) technology and systems may include: often difficult and time-consuming deployment of new resources, quickly changing versions of standards and software, complexity of the middleware for average users.

For integration of Web Services there exist Web Services Flow Language (WSFL), its extension Services Workflow Language (SWFL), BPEL and WS-Choreography.

The J2EE-based BeesyCluster ([6]), implemented by our team, deployed at Academic Computer Center Gdansk, Poland, is an advanced access portal to HPC resources/PCs and services available on them. While the implemented workflow features are similar to other systems, unlike many middlewares BeesyCluster accesses clusters/PCs via SSH. Thus attachment of resources, unlike for many grid systems, can be done in seconds since does not require any additional software layer on clusters/PCs.

[⋆] Partially covered by the Polish National Grant KBN No. 4 T11C 005 25.

[⋆⋆] Calculations carried out at the Academic Computer Center in Gdansk, Poland.

2 Integration of Services into Workflows in BeesyCluster

As an exemplary workflow in BeesyCluster we consider the one in Figure 1. User `iccs` runs a DataGenerator on a Linux workstation `TBC2` (Pentium 4 2GHz) which prepares input data and runs two parallel simulations on two different clusters: `holk` (288 Itanium processors, Infiniband) with PBS and `parowiec` (16 processors, Ethernet) with no queuing system. Applications are offered by different users in BeesyCluster.

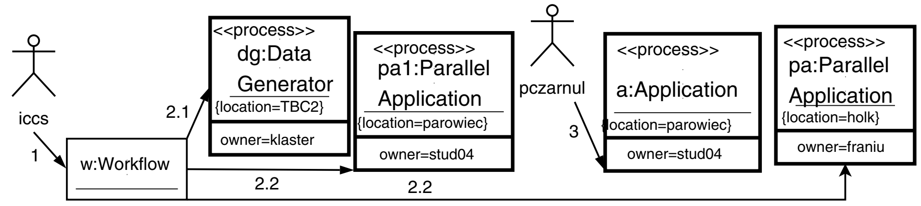


Fig. 1. Basic Interaction Diagram for Testbed Workflow

We have implemented a workflow that responds dynamically to the changes of load on cluster `parowiec` (application `a` started by `pczarnul`), by checkpointing/restarting or migrating `pa1` using our checkpointing libraries for MPI programs ([7]).

In BeesyCluster, services (tasks run interactively, queued or any services that operate on any of resources attached to clusters/PCs) process input data and produce output results. Workflows in BeesyCluster can be modeled as a directed graph with:

node of a workflow graph – contains calls to at least one service. A path of connected graph nodes is implemented as a message-driven bean thread on the J2EE platform calling services on clusters/PCs via SSH. The workflow node is represented by a row in a database with service data. The graph links the node with following ones.

parallel execution of workflow nodes/paths – threads representing parallel paths are created and execute services in parallel.

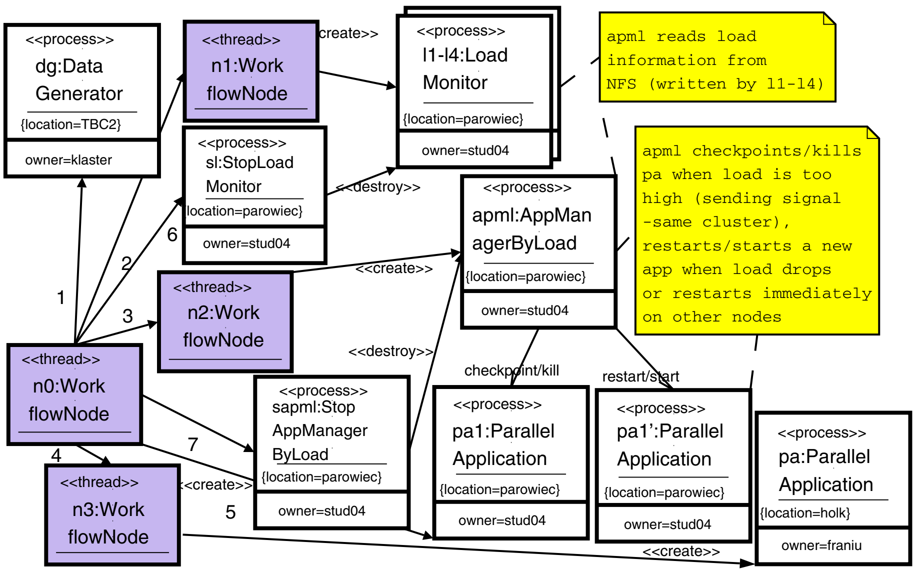
synchronization of parallel workflow nodes on the following node – threads representing parallel paths can determine through synchronization on the database which finishes last and it continues with services associated with the following node.

communication between parallel workflow nodes - inter-thread interaction in the application server or by direct interaction (hand-coded) if nodes on the same resource.

input/output data management – output files of preceding service(s) transferred as input files with proper name mapping.

For the testbed scenario (Figure 1) additional services are created (Figures 2, 3):

1. `l1-l4` on cluster `parowiec` for load monitoring – implemented as a C++ program `loadmonitor` using command `ps` and parsing output periodically,
2. `apml` on cluster `parowiec` – reads the load provided by `loadmonitors`, checkpoints/stops `pa1` when the processor load by other user processes exceeds 10% and restarts when it drops below or migrates to other nodes immediately; `apml` sends signal `USR1` to checkpoint `pa1` and restarts by a system command ([7]).

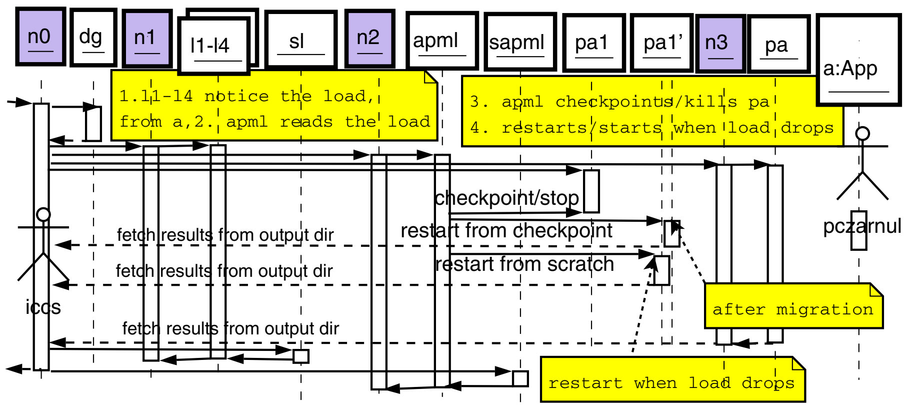


<<process>> dg:Data Generator {location=TBC2} owner=klaster
<<thread>> n1:Work flowNode
<<create>>
<<process>> l1-l4:Load Monitor {location=parowiec} owner=stud04
apml reads load information from NFS (written by l1-l4)
<<process>> sl:StopLoad Monitor location=parowiec} owner=stud04
<<destroy>>
<<process>> apml:AppManagerByLoad {location=parowiec} owner=stud04
apml checkpoints/kills pa when load is too high (sending signal -same cluster), restarts/starts a new app when load drops or restarts immediately on other nodes
1
2
3
6
<<thread>> n2:Work flowNode
<<create>>
<<destroy>>
<<thread>> n0:Work flowNode
<<process>> sapml:Stop AppManager ByLoad {location=parowiec} owner=stud04
checkpoint/kill
restart/start
<<process>> pa1:Parallel Application {location=parowiec} owner=stud04
<<process>> pa1':Parallel Application {location=parowiec} owner=stud04
<<process>> pa:Parallel Application {location=holk} owner=franiu
4
7
5
<<thread>> n3:Work flowNode
<<create>>
<<create>>

Fig. 2. Cooperation Diagram for Testbed Workflow with Reaction to External Load on Cluster

3 Simulations

Figures 3, 4 and 5 present an interaction diagram and results from the execution of the workflow respectively. In the scenario from Figure 4 `pa1` is restarted on the same node when the load from other users drops while for Figure 5 migrated to another node immediately. On `holk` requesting more nodes requires longer to obtain resources (queued via PBS). On `parowiec` tasks are run at once (no queuing) but the cluster has



n0
dg
n1
l1-l4
sl
n2
apml
sapml
pa1
pa1'
n3
pa
a:App
1. l1-l4 notice the load, from a, 2. apml reads the load
3. apml checkpoints/kills pa
4. restarts/starts when load drops
pczarnul
checkpoint/stop
restart from checkpoint
restart from scratch
fetch results from output dir
fetch results from output dir
fetch results from output dir
icos
after migration
restart when load drops

Fig. 3. Interaction Diagram for Testbed Workflow

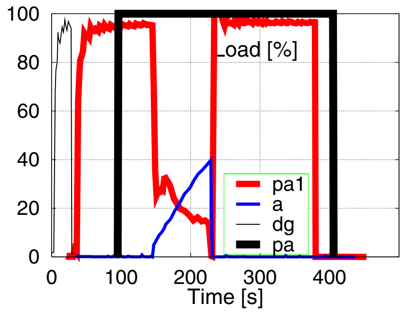


Fig. 4. Workflow with Checkpoint/Restart

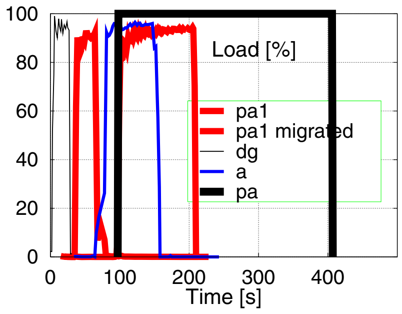


Fig. 5. Workflow with Migration of `pa1`

slower processors and the task can be slowed down by others as in our scenario. For the one in Figure 4 twice as many processors were used on `parowiec` than on `holk`.

4 Conclusions and Future Work

We have presented design and execution of a testbed scientific workflow within BeesyCluster. The workflow responded automatically to launching a higher priority application on cluster `parowiec` by checkpointing/migrating/restarting application. Future work includes incorporation of Web Services and Globus jobs into the system.

References

1. Ludscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger-Frank, E., Jones, M., Lee, E., Tao, J., Zhao, Y.: Scientific Workflow Management and the Kepler System. Concurrency and Computation: Practice & Experience, Special Issue on Scientific Workflows (2005)
2. Deelman, E., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Patil, S., Su, M.H., Vahi, K., Livny, M.: Pegasus : Mapping Scientific Workflows onto the Grid. In: Across Grids Conference, Nicosia, Cyprus (2004) `http://pegasus.isi.edu`.
3. Majithia, S., Shields, M.S., Taylor, I.J., , Wang, I.: Triana: A Graphical Web Service Composition and Execution Toolkit. In: IEEE International Conference on Web Services (ICWS'04), IEEE Computer Society (2004) 512–524 `http://www.trianacode.org/`.
4. Laboratory of Parallel and Distributed Systems, MTA SZTAKI, Hungary: (Parallel Grid Runtime and Application Development Environment, User's Manual, ver. 8.4.2)
5. Foster, I., Kesselman, C., Nick, J., Tuecke, S.: The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. In: Open Grid Service Infrastructure WG. (2002) Global Grid Forum.
6. Czarnul, P., Bajor, M., Fraczak, M., Banaszczyk, A., Fiszer, M., Ramczykowska, K.: Remote task submission and publishing in beesycluster : Security and efficiency of web service interface. In Springer-Verlag, ed.: Proc. of PPAM 2005. Volume LNCS 3911., Poland (2005)
7. Czarnul, P., Fraczak, M.: New user-guided and ckpt-based checkpointing libraries for parallel mpi applications. In Springer-Verlag, ed.: Proceedings of Euro PVM/MPI 2005, 12th European PVM/MPI Users' Group Meeting. Volume LNCS 3666., Sorrento, Italy (2005) 351–358