Characterising and Developing Adaptive Scientific Applications with Hard to Predict Runtime Resource Requirements

Shantenu Jha[1,2,3], Yaakoub El-Khamra[1], Hartmut Kaiser[1], Andre Merzky[1], Ole Weidner[1]

[1]*Center for Computation & Technology, Louisiana State University, USA*

[2]*Department of Computer Science, Louisiana State University, USA*

[3]*e-Science Institute, University of Edinburgh, UK*

May 17, 2008

There are large number of applications that have irregular execution characteristics and highly variable resource requirements which are very difficult to predict in advance. The irregularity may manifest itself in either the total time required to run, time for run for each sub-task, the number of sub-tasks, or irregular communication between the sub-tasks , to name just a few. From the run-time environment perspective, computational Grids are almost by definition, characterised as dynamic and heterogeneous environments. They are dynamic due to time-dependent resource loads, availability and access patterns; the aggregation of specialised resources in different administrative domains is one source of the heterogeneity. We believe that novel, distributed applications must be dynamic and have support for adaptivity – due to either changes in external factors (eg resource availabilty) or due to internal factors (eg more sub-tasks need to be computed). It is important to note, that traditional, monolithic HPC programs are typically not adaptive in response to changing conditions due to either scenario.

In principle there is a difference between "hard to predict" and irregular execution on the one hand and adaptivity on the other. Application-level adaptivity however, is a powerful and simple way to respond to changing resource requirements or availability. But there are many challenges to application level adaptivity – not least of which are a lack of advanced programming models for developing adaptive application, insufficient run-time support for the adaptive aspects and the complexity associated with application-level scheduling.

We have developed and investigate two distinct adaptive applications; both are capable of responding to changing resource availability and performance. In Ref [**?**], we developed an network-intensive application that was able to determine the best run-time configuration based upon bandwidth requirements and network performance. In Ref [**?**], we showed how an application could use computational sensors to determine the best set of resources to launch a varying set of sub-tasks onto. In the latter case, there were additional constraints of global synchronisation and the need to lower the time to completion.

As a consequence of the resource requirements being dynamic and unpredictable, coupled with dependence on the execution trajectory and resources used, it is difficult to define a scheduling strategy that will be effective throughout the complete execution of an applications; hence static resource mapping is generally not an option. It can be argued that the logic for adapting to resource requirement changes from within the application are best addressed at the application level. The constraints and design-space for application-level adaptivity however, are not well understood. In general, such applications are hard to develop and deploy; but in spite of their importance, they have surprisingly received little support for the development and deployment. Also as pointed out in Ref [**?**] it is not easy to programmatically adapt the application performance characteristics so as to be usable by a wide range of application classes.

In this paper, we discuss our approach to the design and development of three adaptive applications – early prototypes that continue to undergo refinement. A common feature in our approach is the use of the the Simple API for Grid Applications (SAGA) [**?**]. SAGA is the first comprehensive attempt to provide a programmatic approach for the development of applications so as to utilize distributed environments – either by design or by virtue of deployment. In addition to simplifying the programming environment for application developers, SAGA insulates applications from technological version changes and other implementation specific details that regularly occur in the lower layers of the software stack.

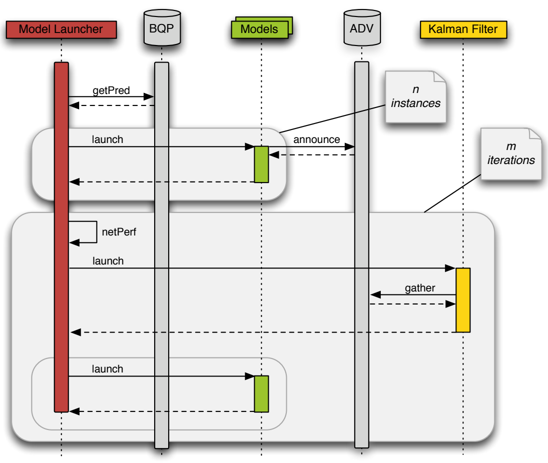

Figure 1: A schematic illustration of control flow of an irregular, hard to predict application, taken from Ref [**?**]. The application is a SAGA-Cactus based ensemble Kalman filter application, where models are generated and these are mapped to appropriate resources. This is done based upon the number of grid points in the model (which in turn determines the number of processors and memory) and the projected run-time (which is based upon estimated number of iterations). An optimal resource – defined as that resource with the highest chances of finishing a job with the prescribed characteristics (number of processors, duration) first is chosen to launch the sub-tasks. After all models (n instances) in a given stage have finished there is a global synchronization point which in turn is the basis for providing models for the next stage (m iterations). It is difficult to determine the values of **n** and **m** *a priori*

Collectively, the applications we investigate span a broad range of the spectrum of *irregular characteristics* that applications have. In addition to the three applications we develop, there exist several other well known examples of applications with irregular run-time characteristics eg., GridSAT [**?**, **?**]. We will use these applications as the basis for defining the core characteristics (vectors) of applications with irregular run-time characteristics and will help define the design space of when adaptivity should be embedded at the application level (eg using SAGA coupled with another framework such as Cactus) versus when it should be handled using system-level frameworks (eg Charm++) or another level altogether.