Perspectives on Multiplicity Computing

Joe Sventek
University of Glasgow
Main Points from Day 1

• There will continue to be an ever-increasing multiplicity of computational resources available for exploitation

• We wish to exploit this multiplicity of (diverse) resources to address functional and non-functional aspects of applications

• Many of the approaches for addressing non-functional aspects require that one be able to generate software variants that are statistically independent

• Genetic algorithms offer one promising technique for generating such variants
Infrastructure Issues

• Whatever approaches/techniques are chosen for exploiting resource multiplicity, significant distributed system platform support is required
• Most use cases include self-configuration, self-optimization, and self-healing
• But wasn’t this promised by “autonomic computing”?
Three general types of “application”

• Explicit application design targeted at resource multiplicity – e.g. MapReduce style processing
• Implicit structure of an application drives exploitation of resource multiplicity – e.g. actor-style application
• Dynamic construction of “applications” to exploit resource multiplicity – e.g. Achieve reliability/dependability through replication
Programming Languages

• Traditional programming languages compile away any notion of components or objects in the source code
• The unit of concurrency is a thread of control
• Dynamic adaptation of an application to resource multiplicity requires that it be straightforward for the infrastructure to determine the concurrent units for redistribution
• It must be straightforward to be able to adapt the unit of concurrency to heterogeneous resources
Actor Languages

• Intimately bind unit of concurrency to data encapsulation
• Interaction between actors is through narrow, strongly-typed communication interfaces
• If the source code is compiled to a common intermediate representation (byte code), provides scope for adaptation to heterogeneous resources (JIT compilation)
What about existing languages

• Develop the toolchain such that structural information available in the original source language that can aid redistribution is maintained; a recent dissertation at ETHZ (one of Gustavo’s students) shows how to do this with Java/OSGi

• Static analysis of other languages?

• Dynamic runtime monitoring to discover units of concurrency?
Knowledge of the resources

• Sophisticated, scalable monitoring of resource availability needed to enable self-configuration, self-optimization, self-healing
• Availability needed at initial configuration time and during runtime
• May be needed at earlier epoch, when dynamically-constructed applications are being assembled
• Must be able to operate at the required scale and enable sophisticated querying of the monitored information
Other concerns

• There will be policies that constrain the composition and adaptation of these applications – how are they specified, who’s responsible for making sure they are enforced?