# Exposing Application Alternatives

**Authors:** Peter J. Keleher, Jeffret K. Hollingsworth, Dejan Perkovic

Department of Computer Science
University of Maryland

## Overview

- Introduction
- Harmony Structure
- Application to system API
- Policies
- Prototype
- An Example Application
- Related Work
- Conclusions

## Introduction

- Meta-computing
- *Active Harmony* architecture
- Tuning Options
- Targets of the *Active Harmony* Architecture
- Principal Questions

## Meta-Computing

- Def: simultaneous and coordinated use of semi-autonomous computing resources in separate locations
  - Efficiency
  - Distribution
  - Heterogenity
  - Changing resources requirements and capacities
**Active Harmony**

- Def: software architecture that manages distributed execution of computational objects in dynamic environments
  - centralized resource manager
    - detailed performance and resource information in one place
    - adaptation of any and/or all applications to improve resource utilization
    - “intra-application” management

**Tuning Options**

- Def: sets of mutually exclusive application configuration alternative
  - Examples
    - number of nodes to be used by application
    - location of query executions (server or client)
    - algorithm tuning (table-driven lookup or sequential search)

**Targets of the Active Harmony Architecture**

- Long-lived applications
  - advantage: amortization of the expensive operations
  - examples: scientific code, data mining
- Persistent applications
  - advantage: higher potential for improvement
  - examples: file servers, information servers, database management systems

**Principal Questions**

- “Can we build an API that is expressive enough to define real-world alternatives?”
- “How can we specify the relationships between the requirements?”
- “Can the Harmony system use this API to improve the behavior of applications during execution?”
Harmony Structure

- Overall Harmony Structure
- Adaptation Controller
- Metric Interface
- Tuning Interface

Adaptation Controller

- Job
  - gather information about applications and environment
  - project the effects of the changes
  - weigh competing costs and expected benefits
- Tuning Criteria
  - network latency and bandwidth
  - memory utilization
  - processor time
  - frictional cost

Metric Interface

- Job
  - providing a unified way to gather data about the performance of applications and their execution environment
Tuning Interface

- **Job**
  - providing method for applications to export tuning options to the system
- **Important Properties**
  - definition of the expected consumption of the resources
  - description of the effects

Application to System API

- Interface between applications and the Harmony adaptation controller used to specify tuning options
- Matchmaker and Globus RSI systems
- Harmony Differences
  - more information
  - resource usage to be quantified
  - better judgement

Matchmaking

- **Components:**
  - classad specification
  - advertising protocol
  - matchmaking algorithm
  - matchmaking protocol
  - claiming protocol

Matchmaking (cont.)

```
Entity (Requestor)

Match Notification (3)

Claiming (4)

Entity (Provider)

Match Notification (3)

Advertisement (1)

Advertisement (1)
```

Match Algorithm (2)
### Capabilities of Tuning Option API
- ability to express mutually exclusive choices
- specification of resource requirements
- relationship between entities
- appropriate granularity
- computation of the frictional cost
- response time

### Harmony Resource Description Language (RSL)
- **Overview**
- **Features**
  - bundles
  - resource requirements
  - performance prediction
  - naming

### Harmony RSL Overview
- **Provides**
  - a uniform set of abstractions and syntax used to express both resource availability and requirements
- **Consists of**
  - set of interface routines
  - a default resource hierarchy
  - a set of predefined tags
- **Is implemented**
  - on top of TCL

### Tags
- harmonyBundle (application bundle)
- node (characteristics: CPU, Memory, OS)
- link (network link characteristics: bandwidth)
- communication (alternative to link)
- performance
- granularity (reconfiguration rate)
- variable
- harmonyNode (resource availability)
- speed (relative to the “sample” CPU)
Tcl

- Features
  - high-level scripting language
  - interpreted
  - extensible
  - embeddable

- Utilization by Harmony
  - resource requirements (Tcl lists)
  - simple incorporation to the application
  - performance is not an issue

Harmony RSL Features

- Bundles
  - Application specifies bundles. Bundle consists of mutually exclusive options

- Resource Requirements
  - High-Level Resources: nodes and communication links
  - Resource Tags: memory amount, CPU cycles

Harmony RSL Features (cont.)

- Performance Prediction
  - throughput (default)
  - response time for individual applications

- Naming
  - flexible and expressive naming scheme
  - naming from within and without
  - fully qualified name:
    - application.instance.bundle.option.resourcename.tagname

Simple Parallel Application

- Generic parallel application running on four processors
harmonyBundle Simple - {
  {- node “worker”
    {hostname “*”}
    {os “linux”}
    {second “300”}
    {memory 32}
    {replicate 4}
    {communication “2 + 2 * 4”}
  }
}
**Variable Parallelism**

- Application implementing “bag-of-task” paradigm
- Each worker process asks for the task from the server, performs it and returns the result back
- Advantages
  - varying parallelism
  - some load-balancing on arbitrary-shaped tasks

```rsl
harmonyBundle bag howMany {
  {default          {node "worker"
    {hostname "*"}
    {os     "linux"}
    {seconds "200/workersNodes"}
    {memory 32}}}
  {variable worker "workerNodes" 1 2 4 8}
  {communication "2 + 2 * workerNodes * workerNodes"}
  {performance [[interp workerNodes {1 1e5} {4 3e4} {8 2e4}]]}
}
```

**Client-Server Database (RSL Code)**

```rsl
harmonyBundle Dbclient:1 where {
  {QS {node server
       {hostname harmony.cs.umd.edu}
       {seconds 9}
       {memory 20}}
    {node client
      {hostname *}
      {os     linux}
      {seconds 1}
      {memory 42}}}
  {link client server 2}}
```

```rsl
{DS {node server
     {hostname harmony.cs.umd.edu}
     {seconds 1}
     {memory 20}}
  {node client
    {hostname *}
    {os     linux}
    {memory >=17}
    {seconds 9}}
  {link client server
    {44 + (client.memory > 24 ? 24 : client.memory) - 17}}}
```
Harmony Policies

- Used by the automatic adaptation system to assign resources to applications
- Components
  - Matching
  - Composition of the global performance information
  - Overall objective function (to be optimized)

Matching Resource Needs

- Harmony StartUp
  - nodes: memory and computing capacity
  - links: bandwidth and latency
- Allocation
  - first-fit strategy
- Matching
  - reduction of the available resources

Explicit Performance Models

- Objective function minimizes the average completion time of the jobs. Application lifetime is needed
- Critical path (inter-process dependencies)
- Future work will be devoted to the improvement of the objective function
  - requirements: single variable, overall good behavior

Setting Application Options

- Too much work to evaluate the performance improvement for the entire space of options combinations
- Greedy strategy: one bundle at a time
  - iterate over list of applications, iterate over a list of options
  - reevaluate once in a while
Harmony Prototype

Harmony Prototype (cont.)

- Harmony process is a server listening on a well-known port
- Harmony-aware applications start, connect to the Harmony server and supply bundles they support
- Applications read special Harmony variables to make run-time decisions
- Harmony Process is an event driven system that waits for the application events

An Example Application

Related Works

- GLOBUS and Legion
  - large projects addressing many different requirements to build a meta-computing environment
  - Harmony: specific problem of developing interfaces and policies
Related Works (cont.)

- Odyssey
  - gives resources to applications on a best-effort basis
  - registers callbacks to notify applications
- EMOP
  - allows applications to define their own load-balancing and communication services
  - provides object migration facilities

Related Works (cont.)

- Condor
  - allocate computing resources to idle nodes
- AppLes
  - presents a list of candidates to be used as a resource
  - application makes its own decision
  - Harmony: overall performance objective, applications say what they need, rather than select from the currently available resources

Related Works (cont.)

- Computational Steering
  - provides a way to alter application’s behavior
  - Main Disadvantage: manual change is needed
- Autopilot (computation steering)
  - applications are adapted automatically
  - Harmony: coordination of the resource utilization by multiple applications

Conclusions

- Harmony as an architecture used for the applications adaptation
- Export of the application options
- Prototype
- Client-Server database application
- Improvement of the overall performance (throughput) of the system