Policy-driven Quality of Service Management

Michael A. Bauer,
Department of Computer Science
The University of Western Ontario
London, Ontario Canada

Overview

- *Research focus: management of distributed systems and applications*
  - Quality of service
  - Resource Management
  - Policy-driven
- *Implications and Relationships to …*
  - Autonomic computing
  - Grid computing
  - Pervasive computing
  - Instrument/sensor distributed systems

Highlights of Research

- Systems management
  - Architecture; resource adjustment algorithms
  - Policy-driven approaches
  - Variety of applications: multi-media, enterprise applications, voice over IP
- Resource management
  - High performance computing environments
- Pervasive environments
  - Location aware
  - RFID

Outline

- Quality of Service
  - A key driver
  - Determines key requirements
- Examples
  - Quality of service management
  - User-driven QoS
- Future

Introduction

- What is Quality of Service (QoS)?
  - Non-functional requirements of an application or system.
  - Can be best defined as the level of satisfaction the user derives from the use of an application.
  - Whether it meets, falls short of, or exceeds user expectations.
- Several factors contribute to this satisfaction:
  - Performance
  - Reliability
  - Availability
  - Others ...

Introduction

- What do our applications and computing environments look like?
  - Environments are growing in size and diversity
  - Applications are becoming larger, more complex, and more time sensitive
  - Emergence of many small devices
- Quality of service is important as these settings are proliferating virtually everywhere
  - In many cases, they are in mission critical roles
- Originally designed to provide functionality - not quality of service
  - Quality of service must be managed!

The Challenge: Resource Management

- Resource management is key to quality of service management.
  - With an infinite supply of all computing resources, we could satisfy the quality of service requirements of all users on all applications.
  - Supply is limited, so we must manage resources.
- Given an application, workload, environment, and quality requirements, how can we determine the resource allocation to this application that will meet these requirements?

Approaches to Resource Management

- Traditional approaches to resource management
  - Uniform service discipline
  - All applications have equal access to resources
- Alternative: service differentiation
  - Manage resources to provide preferential service to certain applications
  - Initial focus on network management
  - Need for end-to-end management
- Approaches
  - *Static*: Allocation based on fixed properties, e.g. application name, user, system
  - *Dynamic*: Allocation done during execution

Differentiated Services: Approaches

- Static approach to management.
  - Typically suffers from either wasting resources or missing quality of service requirements.
  - Unable to cope with the dynamic nature of applications and computing environments
- Dynamic approaches to management to date have also had their limitations.
  - Focus on application adaptations or place adaptation control where it does not belong.
  - Make too many simplifying assumptions.
  - Require too much from users or developers.

Our Approach: Objectives

- Take a dynamic approach to quality of service management.
- End-to-end resource management
- Policy based service differentiation
- Approaches which are independent of application, environment, and resources.
- Efficient.
- Easy to use for application users, administrators, and developers.

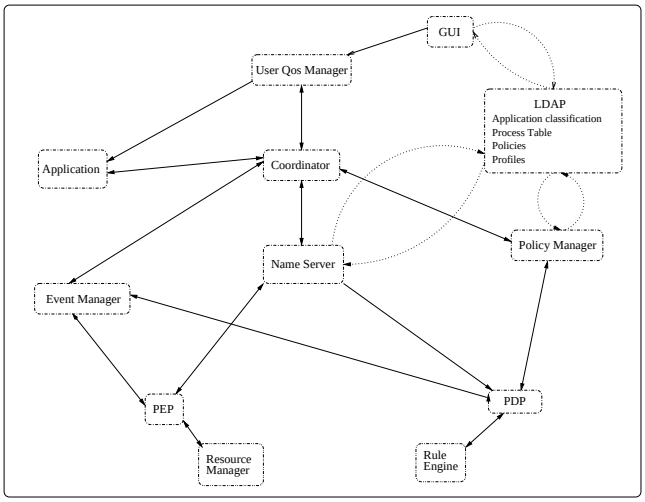
Outline

- Quality of Service
  - A key driver
  - Determines key requirements
- Areas of Study
  - Quality of service management
  - User-driven QoS
- Future

QoS Management

- What are systems and services required for QoS Management?
- How is this done in a distributed environment?
- How are policies enforced?
- What algorithms are needed and what are they?
- ⟶ Development of an architecture of services and components

Architecture



Problem Area: Service Differentiation Policies

- Policies to differentiate based on:
  - Application information: type, name
  - User information: owner of the application, time of execution, application conditions (user intention)
  - Host information: host id, resources
- Policies for *non*-preferred applications
  - Scale back resources
  - Best effort
- Admission control policies
- Challenge: multiple resources
  - Processor, memory, disk, network

Algorithm Development

- Three families of resource management algorithms were developed.
- Metric-heuristic approach
  - Very analytical and diagnostic at run-time.
- Multidimensional search approach
  - Less analysis, performed in advance.
- Round robin approach
  - Very simple, with virtually no analysis whatsoever.

Algorithm Development

- All algorithms have the same basic structure:

```
while application process P is not done do
    Measure quality delivered by P
    if quality does not meet requirements then
        Perform resource management for P
    end if
end while
```

Algorithm Development: Metric-Heuristic Approach

- General idea:
  - Take additional measurements from the application and its environment.
  - Apply heuristics to these metrics to determine which resources to allocate and how much.
- Analysis:
  - Each iteration can be costly to collect metrics and carry out heuristic analyses.
  - If heuristics or metrics are wrong, big trouble.
  - If they are correct, this approach can work well, and can be very adaptive to application or environment changes.

Algorithm Development: Multidimensional Search Approach

- General idea:
  - View the set of all resource allocations as a logical multidimensional search space.
  - Make resource allocations and use resulting quality to update position in the search space appropriately.
  - We need a multidimensional search key though!
  - Use an analysis of variance to split the quality target being searched for into a multidimensional key.
- Analysis:
  - Requires analysis in advance; attempting this analysis at run time is very complex.
  - Several practical limitations in implementation.

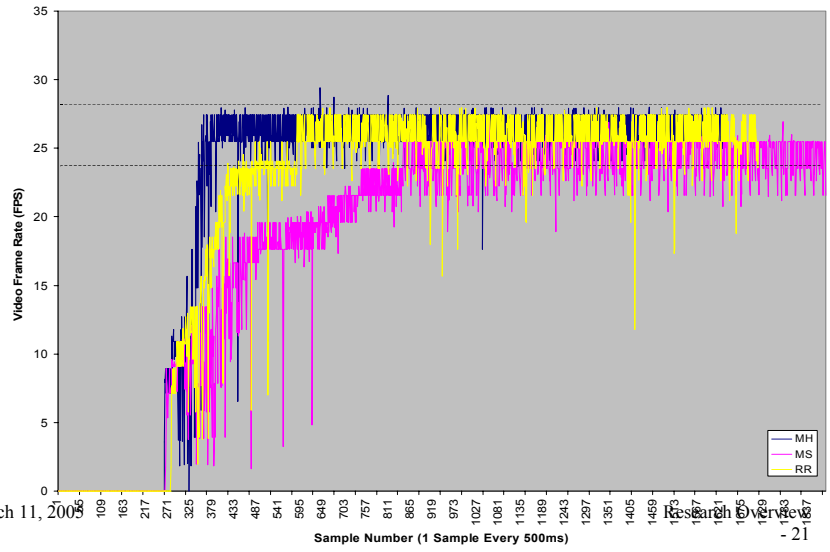
Algorithm Development: Round Robin Approach

- General idea:
  - Iterate through each resource in sequence, allocating each one in an interpolated fashion.
  - Better resource selections can be made by adding weights and dynamically tuning these weights.
- Analysis:
  - Very simple approach, with each iteration of the algorithm being incredibly efficient.
  - Perhaps too simple, and too prone to make wrong resource allocation selections.
  - Using weights may help with minimal additional cost, but they do require prior knowledge.

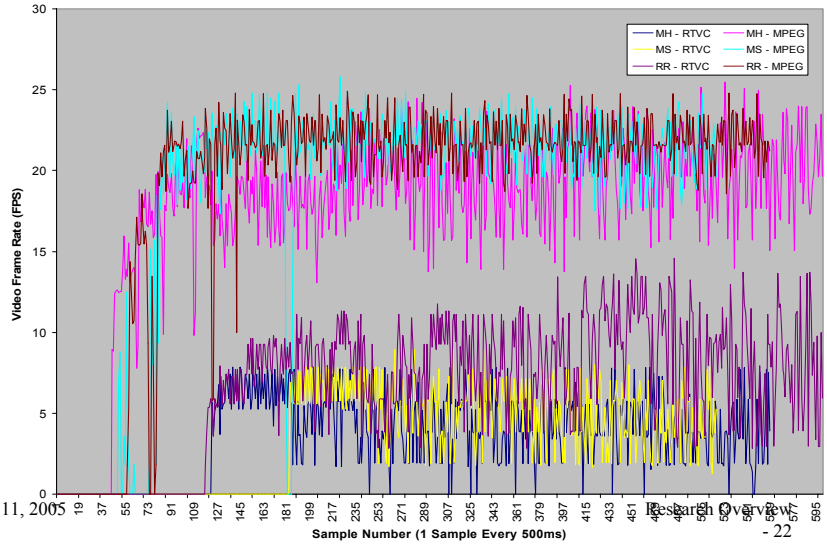
Experimental Results

- Comparison of Algorithms
  - Single application under load
  - Multiple applications under load
- Service differentiation
  - Application: Mpeg viewers (2)
  - Example 1: One Mpeg application is preferred
  - Example 2: One is preferred; change of preference
  - Example 3: Reaction based on user hints

Experimental Results: Single Application Under Load



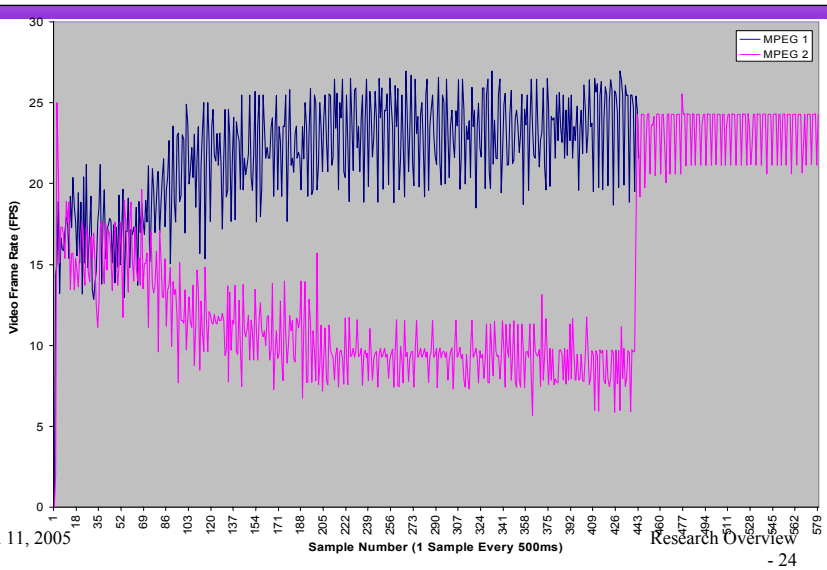
Experimental Results: Multiple Applications Under Load



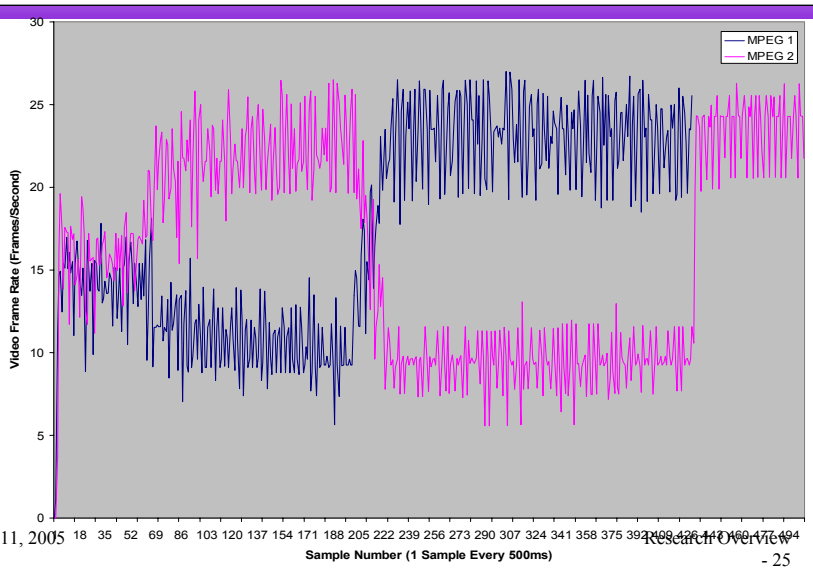
Experimental Results

- Comparison of Algorithms
  - Single application under load
  - Multiple applications under load
- Service differentiation
  - Application: Mpeg viewers (2)
  - Example 1: One Mpeg application is preferred
  - Example 2: One is preferred; change of preference
  - Example 3: Reaction based on user hints

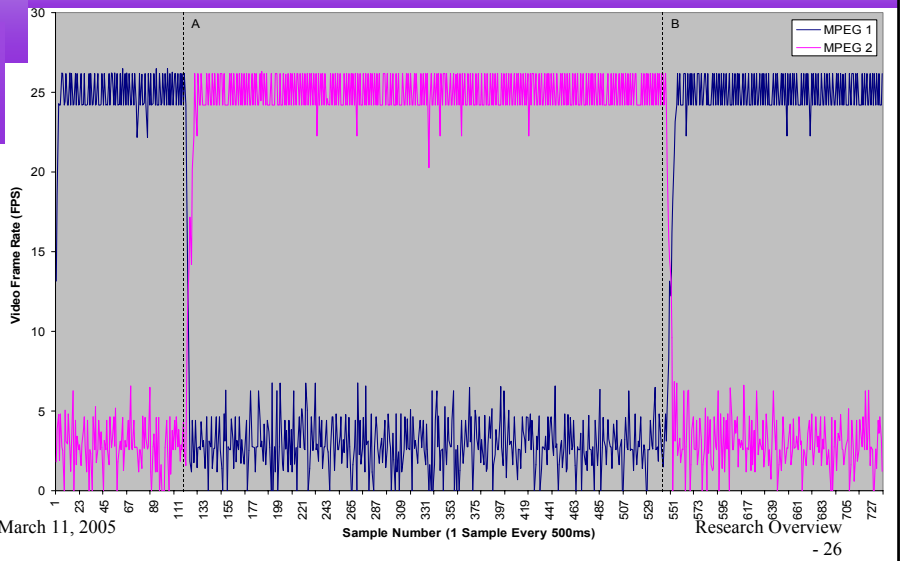
Example 1: MPEG-1 is preferred



Example 2: Changing Preferences During Execution MPEG-2 – then MPEG-1



Example 3: Two MPEG players; User covers and uncovers



Problem Area 2: User-driven QoS

- Reasonable to expect user to influence QoS
  - Not all users will use the same applications and will have different expectations
  - May have different priorities at different times – even for same set of applications
  - Priorities can change unexpectedly
  - Many different applications
- Challenge: enable user to specify preferences and priorities without specific resource requirements

Broad Objectives

- Enable user to dynamically change QoS preferences and priorities among applications
- Minimize technical knowledge required of user
- Support management services to ensure QoS
- Facilitate management by system administrators

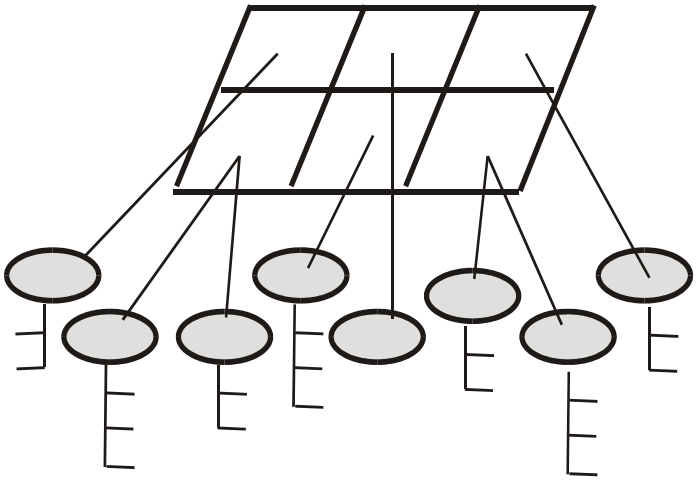
Approach

- Observation: Many applications have similar QoS requirements, e.g. those with streaming video
- Group applications broadly based on service characteristics
- Determine key QoS characteristics for each group of applications
- Develop sets of policies for each group
- Once developed: need only associate an application with a group

Application Classification

- Grouping based on data flows, time constraints and data creation/consumption rates
- *Data flow*: stream, transactional, batch
- *Data creation/consumption*: periodic, aperiodic, random
- *Time*: Time sensitive or not
  - Real time and Non-real time

Categories

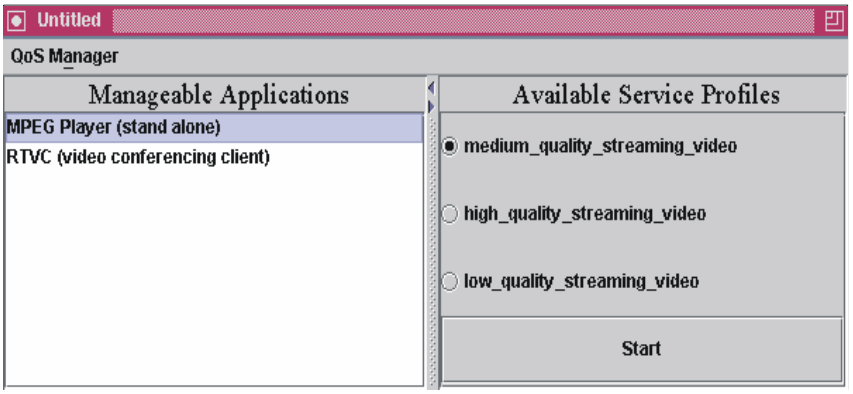


Develop Policies

- Use profiles to develop sets of policies

```
Type oblig client_ Stream_Periodic_RealTime_Video
{ Subject coordinator = /csd/syslab/durendal/coordinator
  Application client = *
    on (client.sensor.current_fps < 20
        and client.sensor.current_fps > 25 );
    do (coordinator.reportViolation(client.sensor.current_fps)}
```

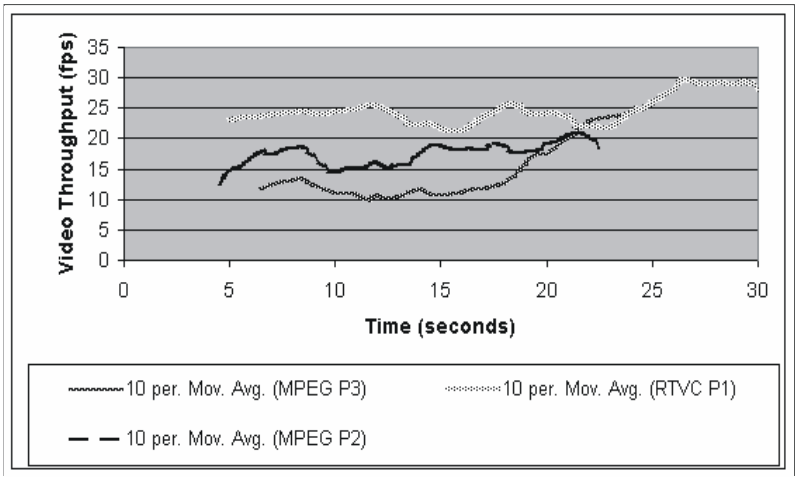
User-Interface Tool



Example

- 2 MPEG applications; 1 RTVC
- All at medium level QoS
- RTVC started first, then the two MPEG
- RTVC at highest priority (user set via the interface)
- Note: running all three simultaneously without QoS management results in all with degraded performance

Example



Future

- Pervasive Computing
  - Deployed in our working and living spaces
  - Context aware and coordinate with each other
  - Quality of service is a key element
  - Manageability is important (especially with large numbers of devices)
- Resource Management in HPC networks
  - Dedicated high bandwidth networks
  - Allocation across clusters
  - Network and storage as resources to be allocated

Questions?