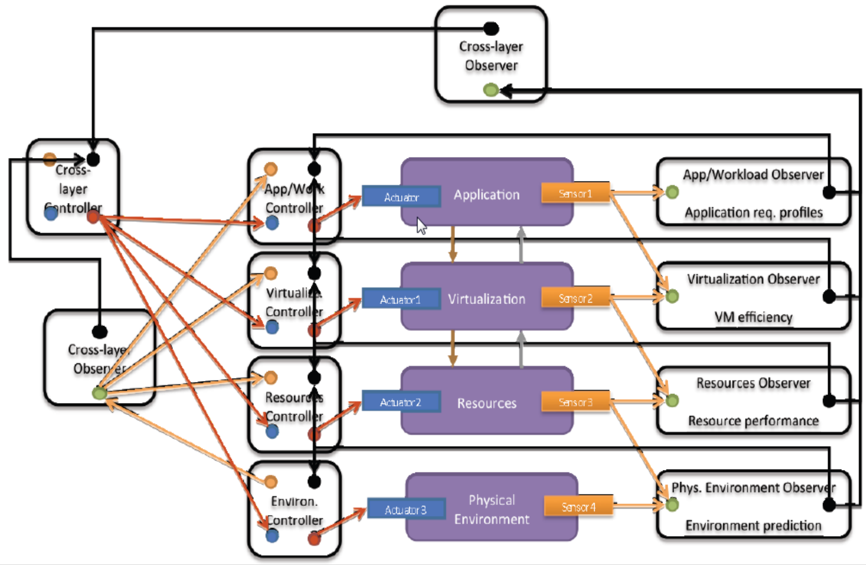
Autonomic Clouds
Part II

Omer Rana (Cardiff University, UK)
Manish Parashar (Rutgers University, USA)

Defining the landscape

- Analysis of existing published work
  - Give potential research directions
- Various uses of autonomics – will focus on:
  - Auto scaling and elasticity
  - Streaming analytics
  - Integrating autonomics into applications
- Integrating autonomics with
  - Existing Cloud middleware
  - Distributed Cloud landscape (e.g. GENI Cloud)

Cross layer Intelligence



Cross layer Intelligence

- Applications can exhibit dynamic and heterogeneous workloads
  – Hard to pre-determine resource requirements
- QoS requirements can differ across multi-tenancy applications
  – Batch vs. real time, throughput vs. response time
- Integrating local resources with Cloud provisioned resources
  – Cloud “bursting” (when and for how long)
  – Data sharing dynamically between the two
  – Cost implications for long term use

From Chenyang Lu (Washington Univ. in St Louis)

Links with Control theory

- Applying input to cause system variables to conform to desired values – often a "set" or "reference" point
  - E-commerce server: Resource allocation? → T_response=5 sec
  - Embedded networks: Flow rate? → Delay = 1 sec
  - Power usage: Energy? → Consumption < 250Watts
- Provide QoS and related guarantees in open, unpredictable environments
- Various modelling approaches:
  - Queuing theory (very popular) – no feedback generally in queuing models; hard to characterise transient behaviour overloads
  - Other approaches: Petri nets and Process algebras
  - Often a design/tune/test cycle – repeated multiple times

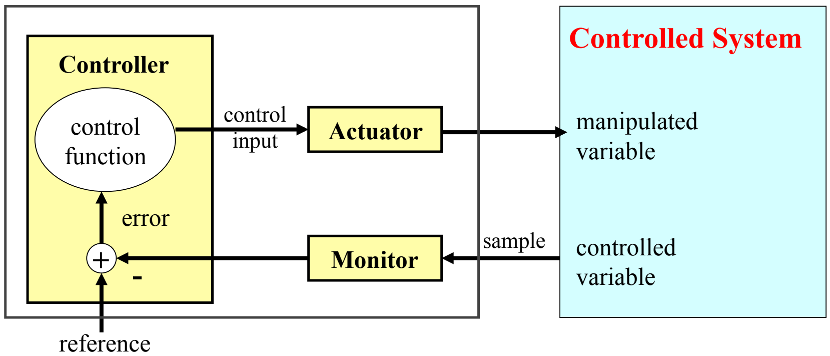
From Chenyang Lu (Washington Univ. in St Louis)

Open-loop control

- Compute control input without continuous variable measurement
  - Simple
  - Need to know **EVERYTHING ACCURATELY** to work right
    - E-commerce server: Workload (request arrival rate? resource consumption?); system (service time? failures?)
- Open-loop control fails when
  - We don't know everything
  - We make errors in estimation/modeling
  - Things change

From Chenyang Lu (Washington Univ. in St Louis)

Feedback (close-loop) Control



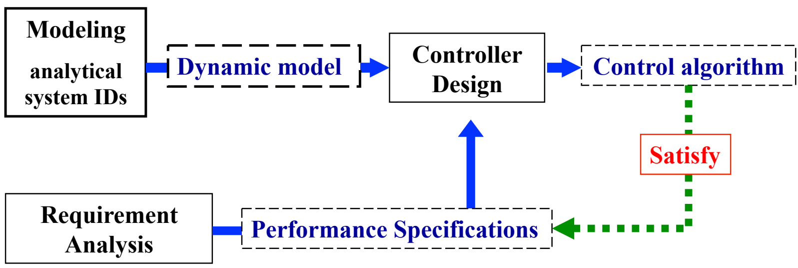
From Chenyang Lu (Washington Univ. in St Louis)

Feedback (close-loop) Control

- Measure variables and use it to compute control input
  - More complicated (so we need control theory)
  - Continuously measure & correct
    - Ecommerce server: measure response time & admission control
    - Embedded network: measure collision & change backoff window
- Feedback control theory makes it possible to control well even if:
  - We don't know everything
  - We make errors in estimation/modeling
  - Things change

From Chenyang Lu (Washington Univ. in St Louis)

Control design methodology



From Chenyang Lu (Washington Univ. in St Louis)

System Models

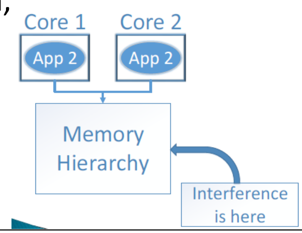
- **Linear** vs. non-linear (differential eqns)
- **Deterministic** vs. Stochastic
- **Time-invariant** vs. Time-varying
  - Are coefficients functions of time?
- **Continuous-time** vs. Discrete-time
- System ID vs. First Principle

- System Goals:
  - Regulation (e.g. target service levels)
  - Tracking (measuring deviation from a target, e.g. change #VMs)
  - Optimisation (e.g. minimize response time)

VM consolidation

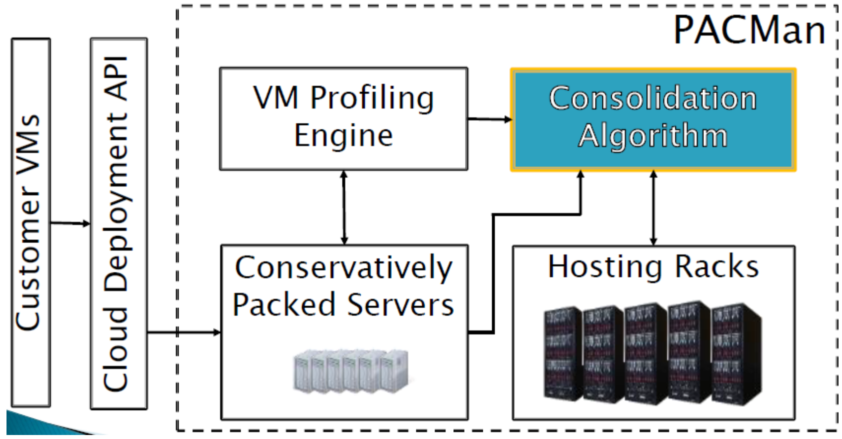
- A commonly referenced problem in Cloud computing
  - Server cost the largest contributor to overall operational cost
- Data centers operate at very low utilization
  - Microsoft: over 34% servers at less than 5% utilization (daily average). US average 4%.
- VM Consolidation increases utilization, decreases idling costs
- However VM consolidation can cause interference in the memory hierarchy

(e.g. due to sharing of cache between cores or memory bandwidth)



VM Consolidation: PACMan

- How much will each VM degrade when placed with other VMs?
- Which and how many VMs can be placed on a server whilst still maintaining performance?
- PACMan (Performance Aware Consolidation Manager)
  - Minimise resource cost (energy usage or #servers)
  - Use of an approximate (computationally efficient) algorithm
- Differentiate between:
  - Performance vs. resource efficiency (e.g. batch mode)
  - Eco mode: fill up server cores (minimise worst case degradation e.g. MapReduce – minimise time of worst case Map task)

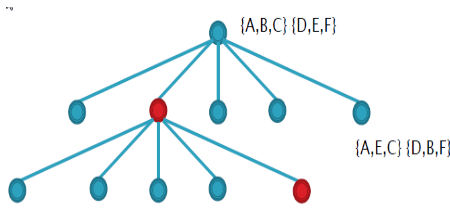


Parameters considered: "n" VMs, on "m" machines with "k" cores
Consider a "degradation" parameter as new VMs are added to a machine – identify an optimisation goal – e.g. Goal: Minimize $\max_{1 \le i \le m} \max_{j \in S_i} d_j^{S_i}$ (S_i is the set of VMs on server i)

Alan Roytman, Aman Kansal, Jie Liu and Suman Nath, "PACMan: Performance Aware Virtual Machine Consolidation", Proceedings of ICAC 2013, San Jose, USA (USENIX/ACM)

PACMan – Approach

- Start from an arbitrary initial schedule
- For all ways of swapping VMs, go to the schedule with smallest sum of maximum degradations
- Limit total number of swaps to achieve convergence



Elasticity

- One of the key "selling points" of Cloud systems
- Various approaches possible:
  - Often historical information useful (response times, queue lengths, arrival rates and request demands)
  - Long-term, medium-term and short-term planning
  - VM allocation and placement
- Reactive vs. proactive approaches

Dynamic VM allocation

- Understanding "Elasticity"

  the degree to which a system is able to **adapt** to **workload changes** by **provisioning** and **de-provisioning** resources in an **autonomic manner**, such that at each point in time the **available resources match** the **current demand** as closely as possible.
- Can elastic provisioning capability be measured

"**Elasticity in Cloud Computing: What It Is, and What It Is Not**"
Nikolas Herbst, Samuel Kounev, Ralf Reussner, ICAC 2013 (USENIX)

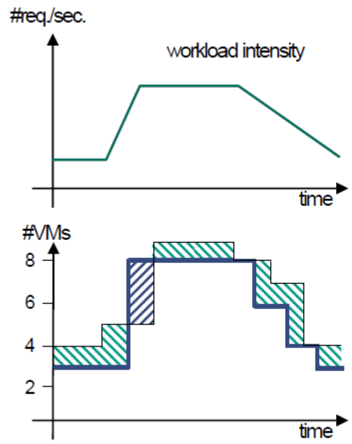
Dynamic VM allocation

- Scale up speed: **switch** from an underprovisioned state **to an optimal or overprovisioned state**.
  - Can we consider "temporal" aspects of how scaling up takes place
- Deviation from actual to required resource demand
  - Measure deviation to influence the overall process
- Role of predictive allocation for "known" events
  - i.e. know in advance how many VMs to allocate

Scaling Influences & Strategies

- Reactive
  - Observed degradation in performance over a particular time window
- Trace-driven
  - Based on a short-term prediction
  - Could make use of a "cyclic" workload pattern
- Model-driven
  - Use of a queuing/Petri net/dynamic systems model
  - Often parameters "observed" and tuned off line

An Example

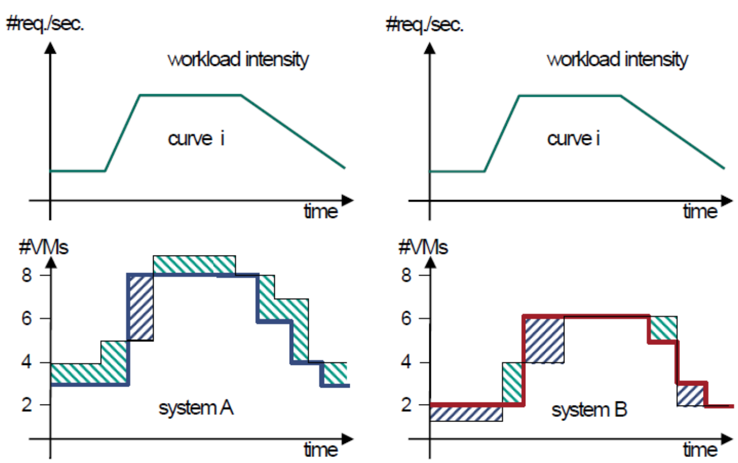


Service Level Agreement (SLA):
E.g.: resp. time ≤ 2 sec, 95%

Resource Demand:
Minimal amount of #VMs required to ensure SLAs.

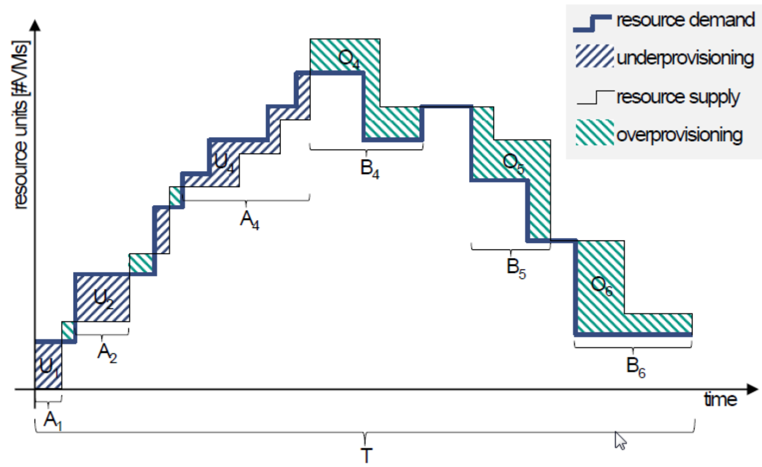
"Elasticity in Cloud Computing: What It Is, and What It Is Not"
Nikolas Herbst, Samuel Kounev, Ralf Reussner, ICAC 2013 (USENIX)

Comparing allocation



"Elasticity in Cloud Computing: What It Is, and What It Is Not"
Nikolas Herbst, Samuel Kounev, Ralf Reussner, ICAC 2013 (USENIX)

Elasticity Metrics



"Elasticity in Cloud Computing: What It Is, and What It Is Not"
Nikolas Herbst, Samuel Kounev, Ralf Reussner, ICAC 2013 (USENIX)

Elasticity Metrics … 2

$\bar{A}$	Average time of switch from an underprovisioned to an optimal or overprovisioned state II_rahmen_neu_folge **average speed of scaling up**
$\sum A$	Accumulated time in underprovisioned state.
$\bar{U}$	Average amount of underprovisioned resources during an underprovisioned period.
$\sum U$	Accumulated amount of underprovisioned resources.
$\bar{B}, \sum B, \bar{O}, \sum O$ correspondingly for overprovisioned states	

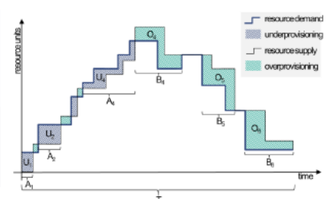
$$P_u = \frac{\sum U}{T}; P_d = \frac{\sum O}{T},$$

$T = total\ evaluation\ duration$

Average precision of scaling up / down

$$E_u = \frac{1}{\bar{A} x \bar{U}}; E_d = \frac{1}{\bar{B} x \bar{O}}$$

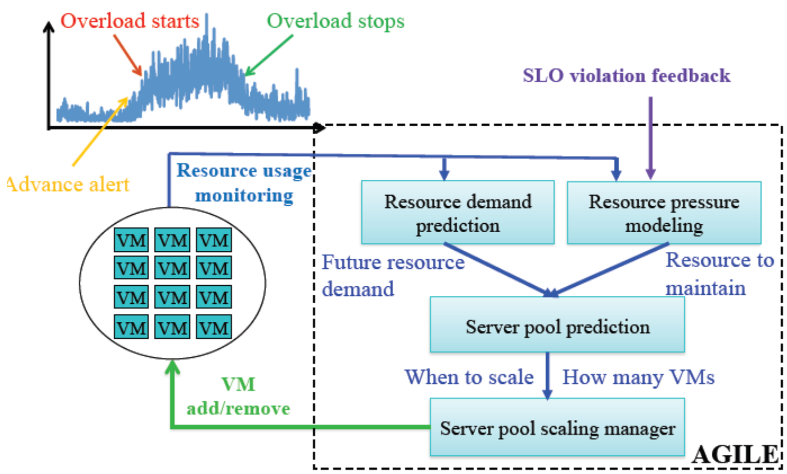
Elasticity metric for scaling up / down



"Elasticity in Cloud Computing: What It Is, and What It Is Not"
Nikolas Herbst, Samuel Kounev, Ralf Reussner, ICAC 2013 (USENIX)

H. Nguyen, Z. Shen, X. Gu, S. Subbiah, J. Wilkes, "AGILE: Elastic distributed resource scaling for Infrastructure-as-a-Service", Proceedings of ICAC 2013, San Jose, USA (USENIX/ACM)

AGILE



AGILE

- Medium term predictions using Wavelets
- Use of an "adaptive" copy rate
  - Pre-copy live VM based on prediction
  - Avoids performance penalty
  - Does not requiring storing and maintaining VM snapshots
  - Can be undertaken incrementally – therefore avoids "bursts" in traffic when submitting an entire VM (e.g. compared to "cold cloning"
- Supports post-cloning auto-configuration

Supporting Elastic Behaviour

- Variety of approaches possible:
- Modelling decisions as a Markov Decision Process (TIRAMOLA successfully resizes a NoSQL cluster in a fully automated manner)
- Use of classifier ensemble
- Machine learning strategies (e.g. use of neural networks)
- Rule-based (trigger-driven) approaches

"**Automated, Elastic Resource Provisioning for NoSQL Clusters Using TIRAMOLA**"
Dimitrios Tsoumakos, Ioannis Konstantinou, Christina Boumpouka, Spyros Sioutas, Nectarios Koziris, CCGrid 2013, Delft, The Netherlands

Hybrid Approaches

- Use of different techniques for scaling up vs. scaling down
  - Reactive rules for scaling up, regression-based techniques for scaling down
  - Reactive rule: queue length of waiting requests (but could be other criteria)
  - Predictive assessment (use of queuing models) to dynamically trigger new VMs

TIRAMOLA

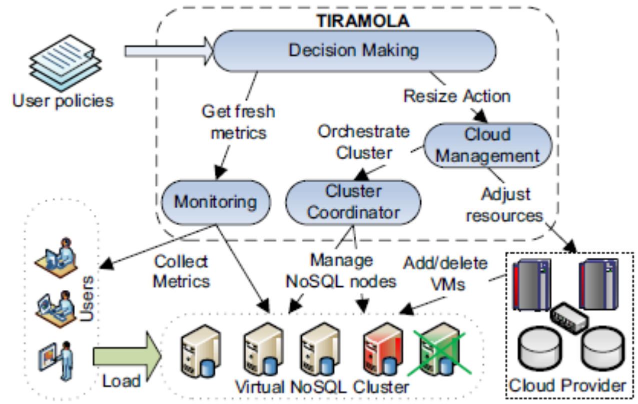


Fig. 1. Architecture of the TIRAMOLA elasticity-provisioning framework.

TIRAMOLA

- Decision Making
  - cluster resize action according to the applied load, cluster and user-perceived performance and optimization policy
  - Modelled as a Markov Decision Process (look for best action w.r.t. current system state)
  - User goals defined through a reward function (mapping of optimisation goals)
- Monitoring via Ganglia
  - Server + user metrics (via gmetric spoofing)
- Cloud Management
  - Via euca2ools (Amazon EC2 compliant REST library)
- Cluster Coordination
  - Via remote execution of shell scripts

TIRAMOLA

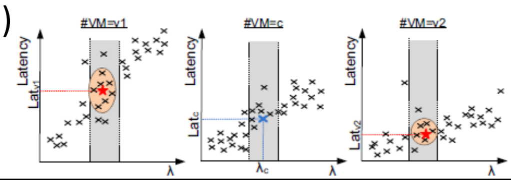
- Formulates resize decisions as a MDP
  - State defined as #VMs, CPU usage, memory
  - Actions: add, remove or do-nothing (no-op)
  - Actions limited by a quantifier, i.e. add_2, add_4 (restrictions on these quantifiers)
  - Transition prob. – based on if state is permissible or not (e.g. can exact number of VMs be added) – can be generalised to partial additions
  - Reward function – r(s): "good ness" of being in state (s); r(s) = f(gains, costs)
- MDP enables:
  - No knowledge of dynamics of environment is assumed
  - Learn in real time (from experience) and continuously during the lifetime of the system

TIRAMOLA

- Use of Q-learning (a type of reinforcement learning)

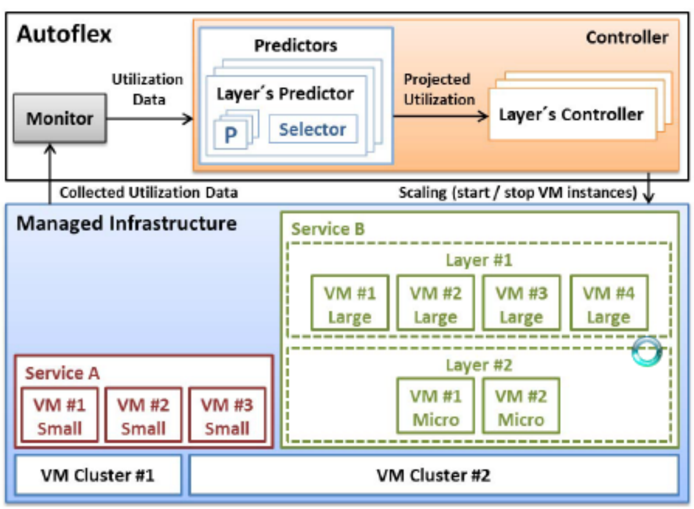
$$Q(s,a) = Q(s,a) + \alpha[r(s') + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

- Base calculation of r(s) on a particular arrival rate (of requests) and certain number of VMs
- Collect results into a table – and use historical data to identify action and s' (given s)
- r(s) = f(latency, VMs)



AutoFlex

- Use of monitoring to collect:
  - CPU, memory, network bandwidth, operating system queues, etc.
- Controller (feedback mechanism)
  - Compares target with actual
  - Launches or terminates VMs
- Controller is both reactive and proactive
  - Layer controllers that run periodically (short term planning)
  - Reactive behaviour through actions for different resource types
  - Predictors attempt to estimate future utilization
  - Multiple predictors – with the use of a selector to choose

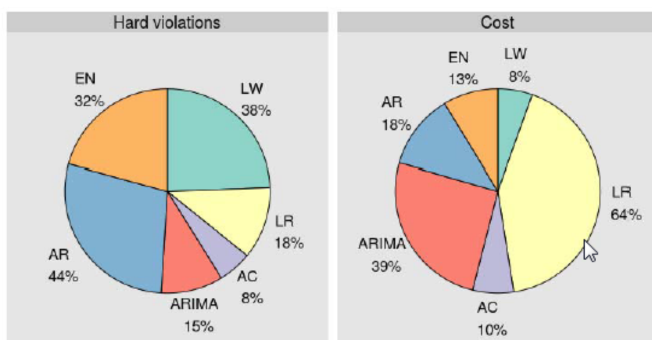


"Autoflex: Service Agnostic Auto-scaling Framework for IaaS Deployment Models"
Fabio Morais, Francisco Brasileiro, Raquel Lopes, Ricardo Araujo, Wade Satterfield, Leandro Rosa
IEEE/ACM CCGrid 2013, Delft, The Netherlands

AutoFlex … predictors

- Keep CPU Utilization < 70%
- Predictors used:
  - auto-correlation (AC),
  - linear regression (LR),
  - auto-regression (AR),
  - auto-regression with integrated moving average (ARIMA), and
  - the previous utilization measured (dubbed Last Window, or simply LW)
  - Ensemble using all of the above
- Metrics:
  - Hard violations: capacity not enough to handle demand
  - Cost: auto scaling vs. over provisioning (knows highest demand and statically allocates resources)

AutoFlex … predictors



- Based on 265 traces from HP users

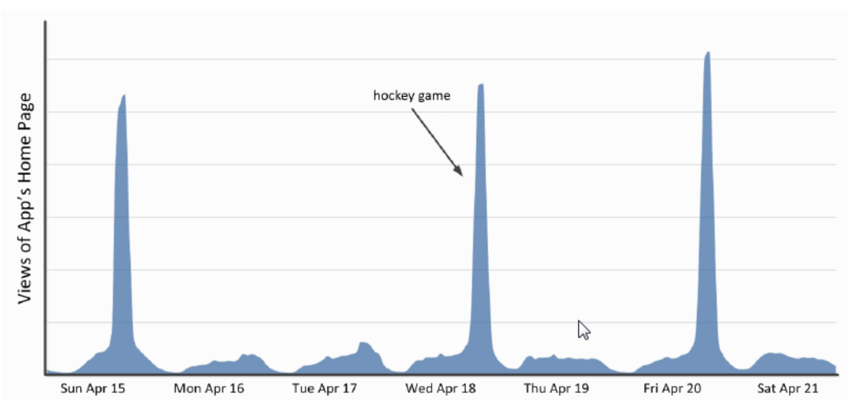
YinzCam (CMU)

YinzCam is a cloud-hosted service that provides sports fans with

- real-time scores, news, photos, statistics, live radio, streaming video, etc.,
- on their mobile devices
- replays from different camera angles inside sporting venues.
- YinzCam's infrastructure is hosted on AmazonWeb Services (AWS) and supports over 7 million downloads of the official mobile apps of 40+ professional sports teams and venues within the United States.

https://www.cmu.edu/homepage/beyond/2008/spring/yinz-cam.shtml

YinzCam – demand profile



week-long workload for a hockey-team's mobile app, illustrating modality and spikiness. The workload exhibits the spikes due to game-day traffic during the three games in the week of April 15, 2012

Auto Scaling strategies

- YinzCam provides an example of various streaming application requirements
- Some events are predictable:
  - Potential workload during a game (historical data) – "in-game" vs. "non-game" mode
  - Some events are not (e.g. likely demand during a particular gaming event)
- Other scenarios:
  - Unpredictable scale up (e.g. observed phenomenon trigger in a sensor network)
- Generally: over provision during game event

Scale up/down policies

- CPU usage threshold → trigger new VM
  - YinzCam (30% CPU usage over 1 minute)
- Aggressive scale up, cautious scale down
  - Overcome VM allocation overheads
  - Potential for oscillation in the system (at next CPU check)
- Example policies:
  - Multiplicative Increase, Linear Decrease
  - Linear Increase, Multiplicative Decrease
- Inspiration from TCP and other congestion control mechanisms

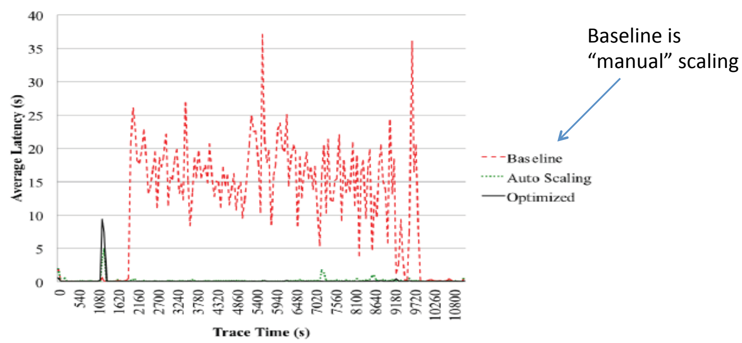


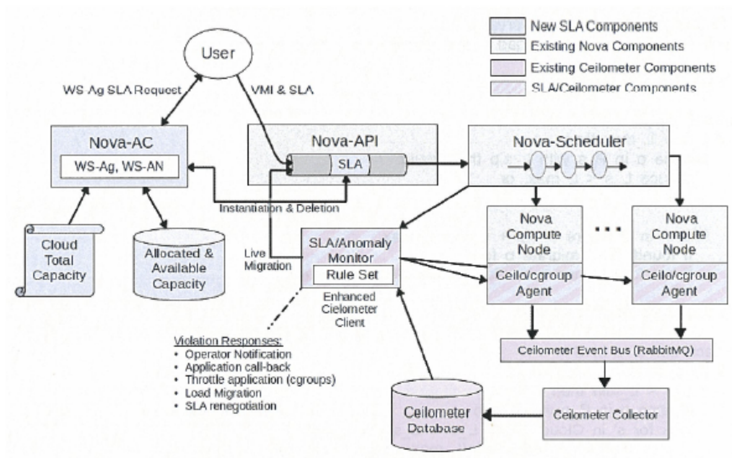
Figure 4: The average latency of our three system configurations throughout a 3-hour production workload trace. The workload was recorded during a hockey-game in April 2012.

"To Auto Scale or not to Auto Scale",
N. D. Mickulicz, Priya Narasimhan, Rajeev Ghandi, ICAC 2013 (USENIX), San Jose, CA

Dynamic SLAs

- Applications on multi-tenancy infrastructure
  - With changing application demands (e.g. must respond to unpredictable events)
- Prevent "over specification" of service level demands
  - User might make an initial assessment of likely demand ("first stab" at likely app. behaviour)
- Provide SLAs that are "machine generated"
  - Based on predictive usage between application classes
  - Offers made to users based on "likely" demand profile
  - May utilise resource throttling strategies (cgroups in Linux – control groups that limit resource consumption)

Dynamic SLAs in OpenStack



"A Research and Development Plan for Dynamic Service Level Agreements in OpenStack",
Craig Lee, ITAAC workshop alongside IEEE/ACM Utility & Cloud Computing Conf, Dresden, December 2013

Admission Control

- Reaching QoS of applications is often strongly driven by admission control strategies
- Admission control in large-scale cloud data centres influenced by:
  – Heterogeneity → performance/efficiency
  – Interference → performance loss from high interference
  – High arrival rates → system can become oversubscribed
- Paragon and ARQ could be two approaches
  – Paragon: heterogenity and interference aware scheduler
- ARQ: Admission control strategy.
  – Use of Paragon to classify applications into multiple request queues
  – Improve utilisation across multiple QoS profiles

"ARQ: A Multi-Class Admission Control Protocol for Heterogeneous Datacenters", Christina Delimitrou, Nick Bambos and Christos Kozyrakis
https://www.stanford.edu/group/mast/cgi-bin/drupal/system/files/2013.extended.arq_.pdf

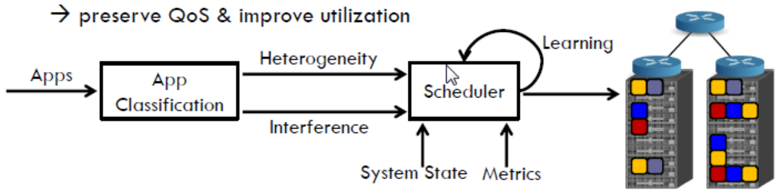
Paragon (Stanford)

- Classification: ~Netflix Challenge
  - Small information signal about new application
  - Leverage system knowledge about previously scheduled applications
  - Collaborative filtering techniques (SVD + PQ reconstruction with SGD)

    → Scheduling recommendations: Heterogeneity + Interference

    Heterogeneity → Server Platform; Interference → Caused (c), Tolerated (t)
- Greedy Scheduler:
  - Co-schedule workloads with no/small interference on suitable hardware platforms

    → preserve QoS & improve utilization



Sources of Interference (SoI) benchmarking

- Targeted microbenchmarks of tunable intensity that create contention in specific shared resources
- Introduce contention in: processor, cache hierarchy (L1/L2/L3 & TLBs), memory (bandwidth and capacity), storage
- Run application concurrently with microbenchmark
  - Progressive tune up intensity until QoS violation
  - Associate a "sensitivity score" with application (i.e. sensitivity to interference)
- Similarly, Sensitivity to running application
  - Impact of running application on micro-benchmark
  - Tuning up application intensity until 5% degradation on benchmark (compared to execution in isolation)

ARQ: Application-aware admission control

- Divide application workload into queues, using
  - Interference tolerance information
  - Heterogeneity requirement
- Trade off between: (i) waiting time; (ii) quality of a resource
- Prevent highly demanding applications from blocking easy-to-satisfy applications
- Understand when a QoS violation is "likely" – re-divert to a different queue
- Interference function (used to derive a resource quality):
  - Interference server can tolerate from the new application (c)
  - Interference new workload can tolerate from existing applications (t)

ARQ: Application-aware admission control

- Resource Quality: Degree of tolerated and caused interference in various shared resources (higher quality means more demanding application)

For application i: $Q_i = \sum_k c_k$ For server j: $Q_j = \sum_k t_k$

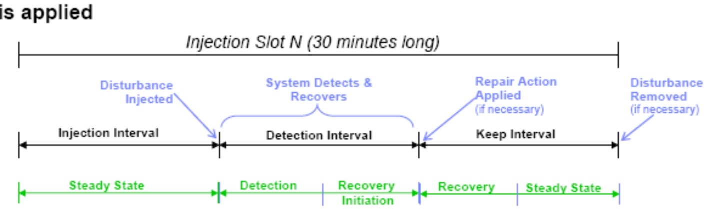
- Resource quality-aware queueing: Applications are queued based on the resource quality they need
- Multi-class admission control: Each class corresponds to apps with specific range of Qi → dispatched to servers with the required Qj
- Preserving QoS: Applications can be diverged to different queues to preserve their QoS (when waiting time is high)

Disturbance Benchmarking

- Tolerance of an application to failure
- Benchmark injects:
  - Workload & Disturbance into System Under Test
  - Measures response
- Disturbance:
  - Events, faults, etc
  - Changes QoS profile of the application
- Aim to measure "resilience" not availability
  - Approach similar to DBench-OLTP
- Ability to adapt in the context of a disturbance in the system

Key Aspect: Injecting Disturbances

- **Each disturbance is injected in an <u>Injection Slot</u> while the workload is applied**



- **Injection slots are run back-to-back, preceded by an optional Startup Interval for ramp-up**
- **For disturbances that require human intervention to recover:**
  - The detection interval is replaced by a fixed, 10-minute time penalty
  - Shorter interval for system that auto-detects but requires manual recovery
  - A scripted Repair Action is applied after the detection interval

From Aaron Brown and Peter Shum (IBM)

Disturbances Injected

- **Benchmark capable of injecting 30 types of disturbances**
  - Representing common expected failure modes, based on internal expertise, data, and customer survey
- **Disturbance types**
  - **Attacks** (e.g. runaway query, load surge, poison message)
  - **Unintentional operator actions** (e.g. loss of table/disk, corrupted data file)
  - **Insufficient resources / contention** (e.g. CPU, memory, I/O, disk hogs)
  - **Unexpected shutdowns** (e.g. OS shutdown, process shutdown)
  - **Install corruptions** (e.g. Restart failures on OS, DBMS, App Server)
- **Targeted at OS, all middleware and server tiers, and application**

From Aaron Brown and Peter Shum (IBM)

Top Customer Pains Overall

Customer Pain
Hang failure of a server: database (DBMS)
Application-related hangs: internal application hang
Leaks: memory leak in user application
Database-related data loss: storage failure affecting database data
Restart failure of operating system on: database (DBMS) node
CPU resource exhaustion on: database (DBMS) node
Miscellaneous hang failures: hang caused by unavailability of remote resource (e.g., name/authentication/directory server)
Miscellaneous Restart Failures: orphaned process prevents restart
Restart failure of server process for: database (DBMS) node
Restart failure of operating system on: application server node
Surges: load spike that saturates application
Miscellaneous stops: Unexpected stop of user application
Database-related data loss: loss of an entire database file
Application performance affected due to: parameter setting on database

Useful to compare this with performance benchmarks that we are much more aware of

Compare with automated testing mechanisms

From Aaron Brown and Peter Shum (IBM)

Metrics for Quantifying Effects of Disturbances (1)

- **Metric #1: Throughput Index**
  - Quantitative measure of Quality of Service under disturbance
  - Similar to typical dependability benchmark measure
  - Computation for disturbance i:

$$ThroughputIndex_i = P_i / P_{base}$$

where

P_i = # of txns completed without error during disturbance injection interval i
P_{base} = # of txns completed without error during baseline interval (no disturbance)

  - Range: 0.0 to 1.0
    - Anything below 0.9 is pretty bad
  - Average over all disturbances to get final score

From Aaron Brown and Peter Shum (IBM)

Metrics for Quantifying Effects of Disturbances (2)

- **Metric #2: Maturity Index**
  - Novel, qualitative measure of degree of Autonomic capability
  - Each disturbance rated on 0 – 8 point scale aligned with IBM's Autonomic Maturity model

Maturity Level	Brief Description	Points
Basic	*IT staff relies on reports, docs, and manuals to manage individual IT components*	0
Managed	*IT staff uses management tools providing consolidated IT component management*	1
Predictive	*Components monitor and analyze themselves and recommend actions to IT staff*	2
Adaptive	*IT components monitor, analyze, and take action independently and collectively*	4
Autonomic	*IT components collectively & automatically self-manage according to business policy*	8

    - Non-linear point scale gives extra weight higher maturity
  - Ratings based on 90-question survey completed by benchmarker
    - Evaluate how well the system detects, analyzes, and recovers from the failure
    - Example: for abrupt DBMS shutdown disturbance:
      "How is the shutdown detected?
        A. The help desk calls operators to tell them about a rash of complaints (0 points)
        B. The operators notice while observing a single status monitor (1 point)
        C. The autonomic manager notifies the operator of a possible problem (2 points)
        D. The autonomic manager initiates problem analysis (4 points)"
  - Overall score: averaged point score / 8
    - Range: 0.0 to 1.0

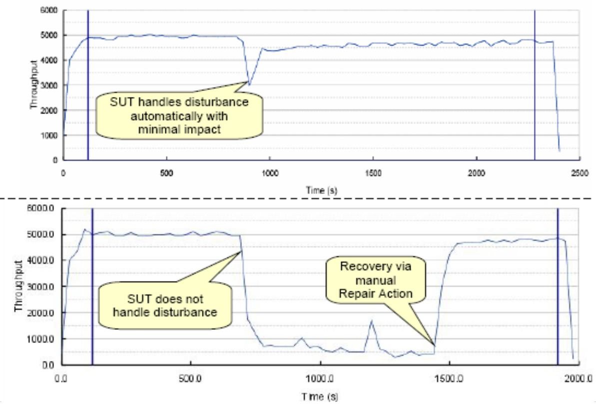
From Aaron Brown and Peter Shum (IBM)

Example Results: Detailed Disturbance Response

- **Comparison of throughput over injection slot for 2 disturbances:**

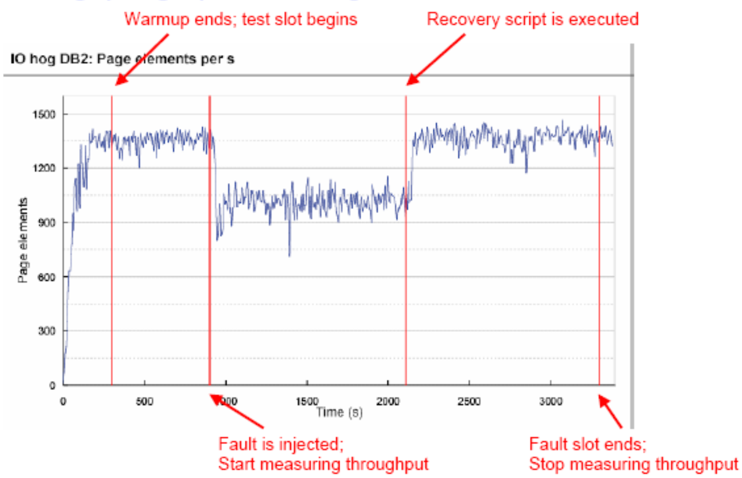
Disturbance #1
(OK response)

Disturbance #2
(poor response)



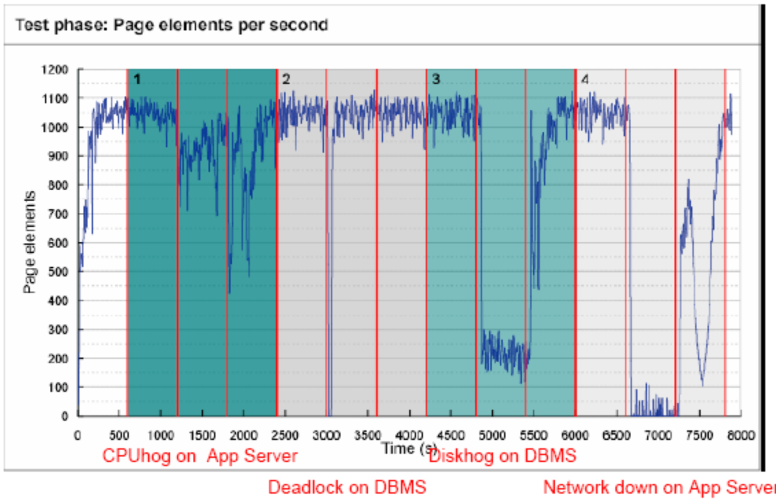
From Aaron Brown and Peter Shum (IBM)

Sample throughput graph for a single fault



From Aaron Brown and Peter Shum (IBM)

Sample graph showing throughput across all four faults



From Aaron Brown and Peter Shum (IBM)

Configuration Management

- Dynamically deploy pre-configured virtual machine instances
  – Replicate across multiple servers
  – Deploy a "reference" configuration across clients
- CHEF – widely used configuration management tool (SaaS platform, Ruby-based)
  – Deploy load balancers, monitoring tools (Nagios) along with others (sharing "cookbooks" and "recipes")
  – Apache Licence (with Apache SOLR (search engine), CouchDB)
- CF Engine
  – Open source (GPL Licence)
  – Enables much more complex configurations (-ve)
  – Uses a remote agent (also supports a monitoring deamon)

http://www.slideshare.net/jeyg/configuration-manager-presentation

Configuration Management

- Amazon CloudFormation another option
  - Create & manage AWS instances -- http://aws.amazon.com/cloudformation/
  - Provides pre-defined set of templates (WordPress, Joomla, Windows Server, Ruby on Rails, etc) -- http://aws.amazon.com/cloudformation/aws-cloudformation-templates/
- CloudSoft's Brooklyn http://www.cloudsoftcorp.com/communities/
  - Open source + support for policies
  - Application-level rather than instance-level support
  - Enables autonomic adaptation of a deployed configuration (e.g. auto-scaling policy, replacer/restarter (high availability) policy)

Stream processing architectures

- Systems that must react to streams of data produced by the external world
- Stream data source can vary in complexity and type
- Availability of streamed data can also be managed through an access control mechanism
- Usually operate in real time over streams and generate in turns other streams of data enabling:

(i) passive monitoring: what is happening, or

(ii) active control: suggesting actions to perform, such as by stock X, raise alarm Y, or detected spatial violation, etc.

- Stream processing can also lead to semantic annotation of events

Difference from "standard" databases

- Queries over streams are generally "continuous"
  - executing for long periods of time
  - return incremental results
  - permanently installed – i.e. does not terminate after first execution
  - Performance metrics should be based on response time rather than completion time
- Data is not static – as new data is constantly arriving into the system
  - Same query at different times leads to different results (as long as new data enters the system)
- Typical operations in StreamSQL
  - SELECT (execute a function on a stream) and WHERE (execute a filter on a stream) operators
  - Stream merge and join
  - Windowing and Aggregation

Analyses of performance

- Response Time
  - Average or maximum time between input arrival into the system, and the subsequent generation of a response
- Support (query) Load
  - What is the size of the input (number of data elements) a stream system can process while still meeting specified response time target and correctness constraints
- Correctness is time dependent
  - Same query at different times → different outcomes
  - Potentially multiple correct answers depending on response time

Adaptive Streams

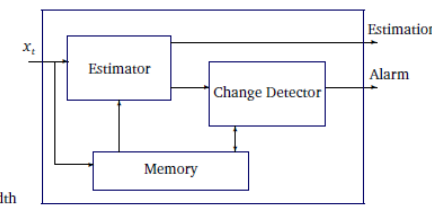
- Three key issues:
  - what to remember or forget,
  - when to do the model update, and
  - how to do the model update
- For streaming – these can be mapped into:
  - the <u>size of the window</u> to remember recent examples
  - methods for <u>detecting distribution change</u> in the input
  - methods for <u>keeping updated estimations for input statistics</u>

All "x" are real valued, estimator: current value of "x" + variance (each "x" independently drawn)

Estimator: linear, moving average, Kalman filter

$$\hat{x}_k = (1-\alpha)\hat{x}_{k-1} + \alpha \cdot x_k.$$

The linear estimator corresponds to using $\alpha = 1/N$ where N is the width of a virtual window containing the last N elements we want to consider.



Adaptive Sliding Windows (ADWIN)

- Window size – reflects time scale of change
  - <u>Small</u>: reflects accurately the current distribution
  - <u>Large</u>: many examples are available to work on, increasing accuracy in periods of stability
- Window content is used for
  - detecting change (e.g., by using some statistical test on different sub windows),
  - to obtain updated statistics from recent examples,
  - to have data to rebuild or revise the model(s) after data has changed
- Adaptive Windowing:
  - Whenever two "large enough" sub windows of *W* exhibit "distinct enough" averages → corresponding expected values are different → drop older portion of window

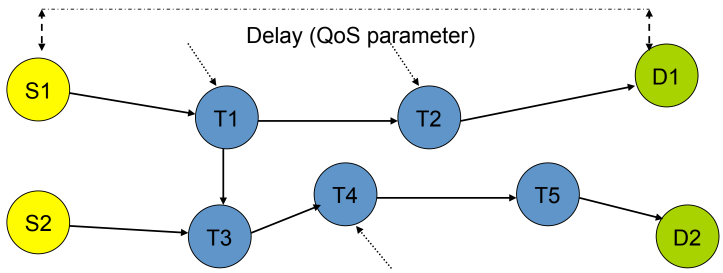
Cloud-based stream processing

- Use of Cloud resources to:
  - Execute stream processing operators (may be in-network)
  - VM per operator (dynamically allocated to overcome peak workloads)
- Operator chaining within/across Cloud systems
  - Scale out
  - Fault tolerance
- Operator chaining → processing pipelines
  - Similarity with workflow systems

GENI (OpenFlow and MiddleBox)

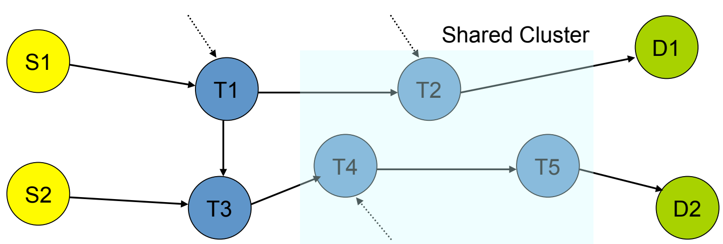
- L2/L3 Technology to permit software-defined control of network forwarding and routing
- Integration of specialist network "appliances" to support specific functions
  - These could be user defined
  - Linux hosting
- MiddleBox: In-network general-purpose processors fronted by OpenFlow switches
- Integrate services from multiple Clouds
  - Allocation of networks and "slices" across different resources

In-transit Analysis

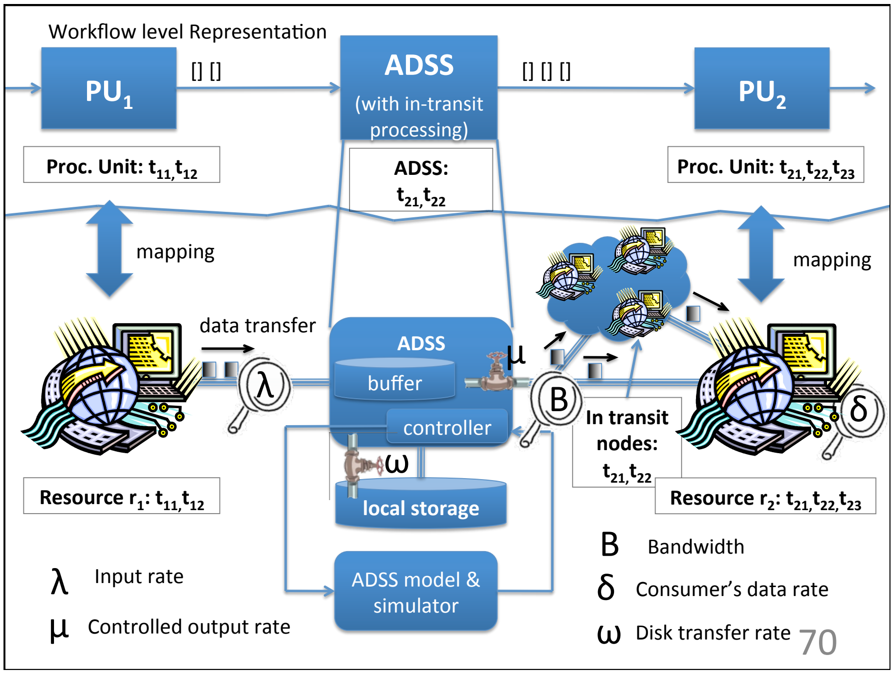


- Data processing while data is in movement from source to destination
- Question: what to process where and when
- Use of "slack" in network to support partial processing
- Application types:
  - Streaming & Data Fusion requirement

In-transit Analysis … 2



- Data processing while data is in movement from source to destination
- Question: what to process where and when
- Use of "slack" in network to support partial processing



Rafael Tolosana-Calasanz et al. "Revenue-based Resource Management on Shared Clouds for Heterogenous Bursty Data Streams", GECON 2012, Springer

Approach & focus

Adaptive infrastructure for sensor data analysis

- **Multiple** concurrent data streams with SLA
- **Variable** properties: rate and data types; various processing models
- Support for **in-transit** analysis, enforcing QoS
- Support for **admission** control & flow **isolation** at each node
- In case of QoS violation, **penalisation**

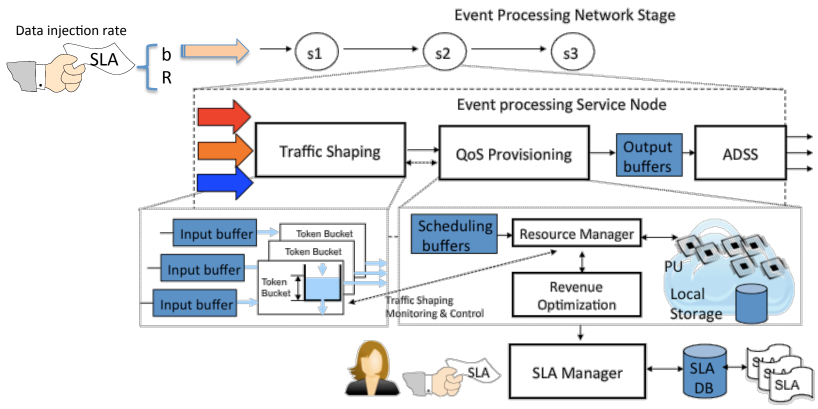
Key focus

- **Architectural components**
- **Business rules** for **SLA** Management **:** **Actions** to guarantee **QoS** & **maximize revenue**

From Jose Banares (University of Zaragoza)

From Jose Banares (University of Zaragoza)

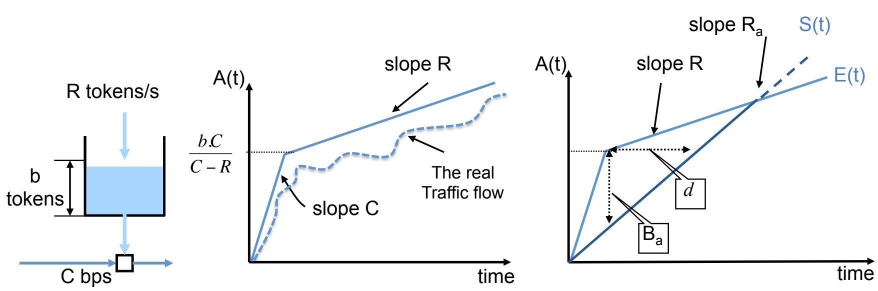
System Architecture



- 3 key components / node: Token Bucket, Processing Unit & output streaming

From Jose Banares (University of Zaragoza)

Token Bucket (shaping traffic)

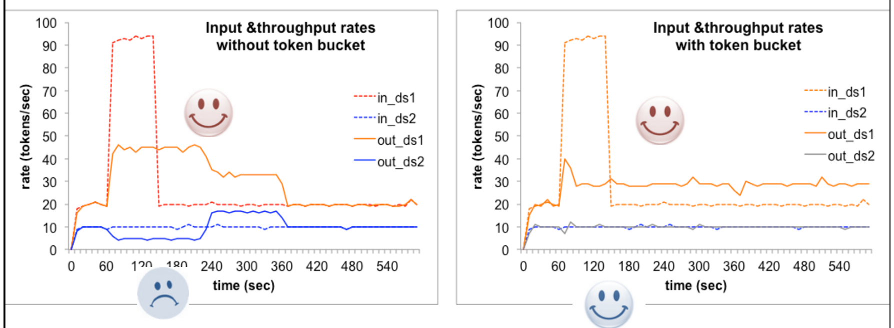


A(t): Amount of data arriving up to time t

Two key parameters of interest:

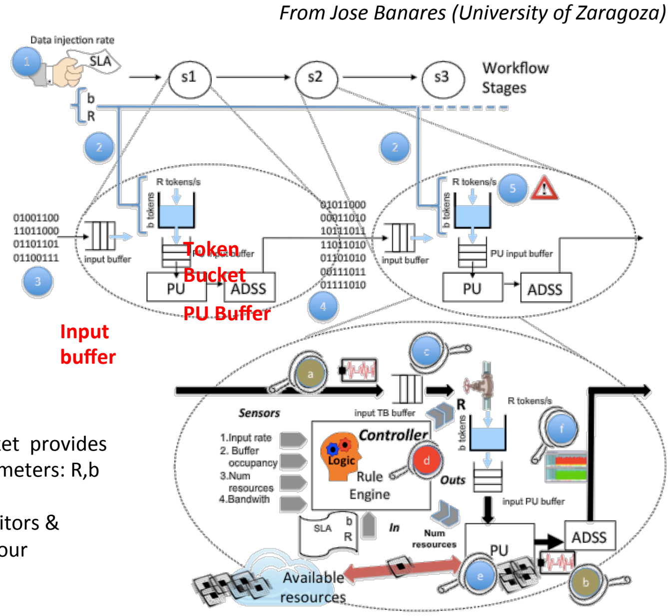
- **R**: Also called the **committed information rate** (CIR), it specifies how much data can be sent or forwarded per unit time on average
- **B**: it specifies **for each burst how much data can be sent** within a given time without creating scheduling concerns

Token Bucket (shaping traffic)



From Jose Banares (University of Zaragoza)

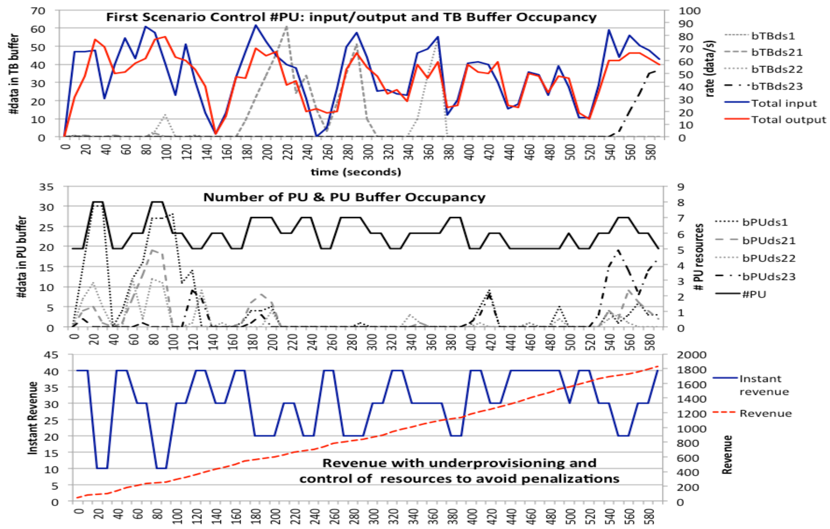
From Jose Banares (University of Zaragoza)



Each token bucket provides us **tunable** parameters: R,b

Controller: monitors & modifies behaviour

Resource addition based on buffer occupancy



Autonomic Computational Science

- **Enable automated tuning of application behaviour**
  - Execution Units, Communication, Coordination, Execution Environment
  - Relation to "Reflection" and Reflective Middleware + Use of intercession on a meta-model + domain-model
  - Developing a meta-model is often difficult
- **Tuning may be:**
  - Centralized
  - Consist of multiple control units
  - Tuner external to the application
- **Comparison with Control systems & MDA**
  - Multiple, often "hidden" control loops
  - Inclusion of run-time configuration parameters at design time
  - Model centric view that takes deployment and execution into account

Shantenu Jha, Manish Parashar and Omer Rana,
Self-adaptive architectures for autonomic computational science
Proceedings of the First international conference on Self-organizing architectures , pp 177-197, LNCS 6090, Springer Verlag 2010.

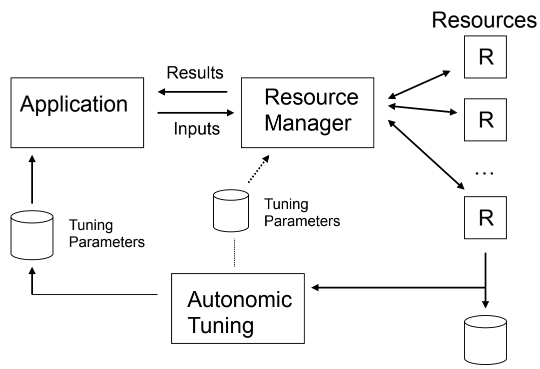
Autonomic Computational Science

Conceptual Framework

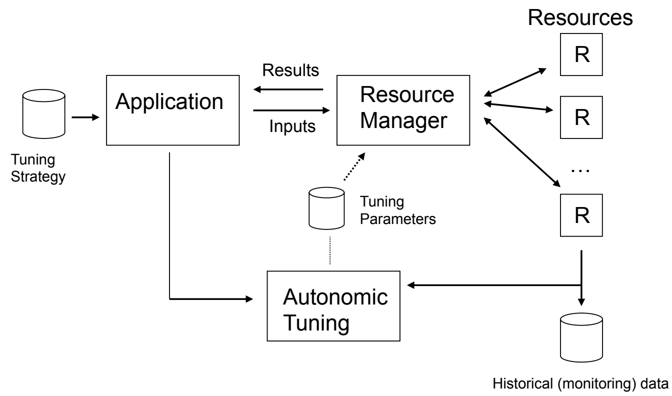
A conceptual framework that comprises of the following elements:

- Conceptual Architectures
- Elements of the Architecture
  - Application-level Objective(s)
  - Mechanism
  - Strategy
- Use in applications driven by the following questions:
  - Which strategy is best for a given application objective? What role do application characteristics play in determining such a strategy?
  - Which mechanism can be used to implement autonomic behaviour – and at which part of the application lifecycle?
  - What support and implementation tools can be used to achieve this autonomic behaviour – and can these be shared across applications?

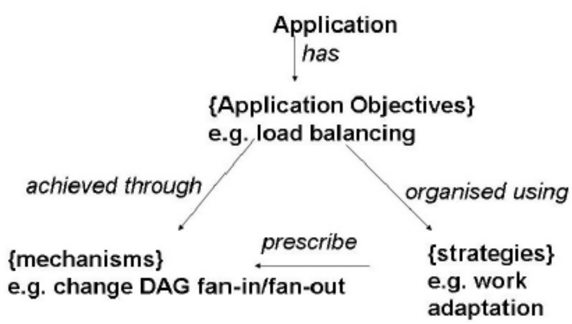
Tuning of application & resource manager parameters



Tuning by application

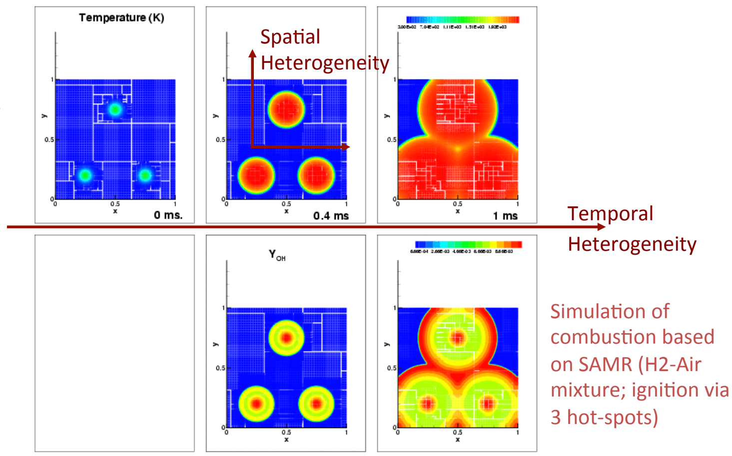


- Resource reservation to achieve particular QoS-criteria
- Dynamic analysis of data stream from a scientific instrument – may also involve analysis of video/audio feeds



- Application-level Objective (AO): User identified application requirement, e.g. increase throughput, reduce task failure, load balance, etc
- Mechanism: action used by application or resource manager to achieve AO – mechanism m: ($\{m_i\}$, $\{m_i^e\}$, $\{m_o\}$, $\{m_o^e\}$), e.g. file staging:
  - $\{m_i\}$ and $\{m_o\}$: file references before/after staging process
  - $\{m_i^e\}$: input events that trigger start of file staging
  - $\{m_o^e\}$: output events after file staging is completed.
- Strategy: consists of a collection of mechanisms – manual or dynamically constructed by an autonomic approach

Spatial, Temporal and Computational Heterogeneity and Dynamics in SAMR



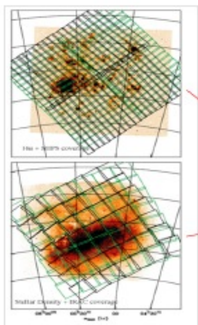
Courtesy: Sandia National Lab

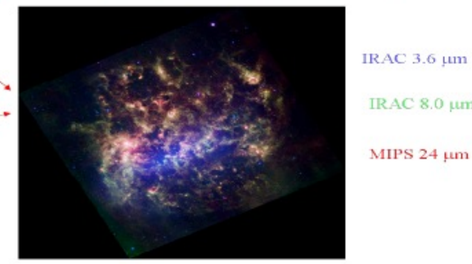
Autonomics in SAMR

- Tuning by the application
  - Application level: when and where to refine
  - Runtime/Middleware level: When, where, how to partition and load balance
  - Runtime level: When, where, how to partition and load balance
  - Resource level: Allocate/de-allocate resources
- Tuning of the application, runtime
  - When/where to refine
  - Latency aware ghost synchronization
  - Heterogeneity/Load-aware partitioning and load-balancing
  - Checkpoint frequency
  - Asynchronous formulations

Montage

Instrument	Bands (μm)	Field-of-View (arcmin)
IRAC	3.5, 4.5, 5.8, 8.0	5.2. x 5.2
MIPS	24	5.4 x 5.4
	70	5.25 x 2.6
	160	0.5 x 0.5

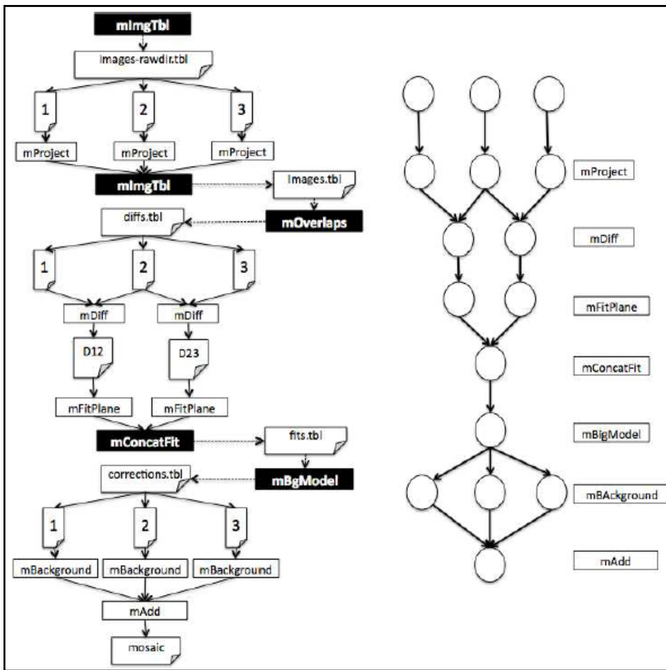




Two epochs:
Jul/Aug 05 & Oct/Nov 05

From: G. Bruce Berriman

Images Courtesy Margaret Meixner (PI)



Montage workflow

I/O intensive workflow

Significant data parallelism

Montage: Tuning Mechanisms

Vectors	Mechanisms
Coordination	Adapt DAG structure, Change Fan In/Out, Cluster Nodes, Change Task Granularity
Communication	File staging, File aggregation, File splitting, File indexing
Execution Environment	DAG execution (Mapping/Scheduling), Resource Selection/Management, Task re-execution, Task migration, Storage management, File caching, File distribution, (multicast, broadcast), File re-transmission, Checkpoint/restart

Montage: Tuning Strategy

Application Objective	Autonomic Strategy
Load Balancing	**1. Adapt task mapping granularity based on system capabilities/state** File staging, File splitting/merging Task rescheduling, Task migration File distribution and caching, Storage Management **2. Change fan-in/fan-out** DAG structure modification File staging, File splitting/merging Task rescheduling, Task migration File distribution and caching Storage Management

Montage: Tuning Strategy

Application Objective	Autonomic Strategy
Handling Task Failure	**1. Reschedule the task on a different existing resource** File staging Task rescheduling, Task migration **2. Reschedule the task on a new resource** Resource discovery and allocation Task rescheduling File staging (migration/replication) **3. Roll back from checkpoint on the same resource** Checkpoint interval and granularity

Montage: Tuning Strategy

Application Objective	Autonomic Strategy
Improving Throughput	**1. Increase fan out** Task rescheduling, Task migration File staging, File splitting/merging DAG structure modification File distribution and caching, Resource allocation **2. Change Scheduling Approach** File distribution (staging, merging, splitting, replication) Task rescheduling and mapping

Concluding comments

- Autonomic strategies:
  - Often rooted in control systems (generally closed-loop feedback control)
  - Can use a variety of control strategies – which include use of machine learning
- Formulating the problem often difficult
  - Multi-criteria optimisation
  - Often multiple, difficult to separate control loops
- Monitoring infrastructure choice is key