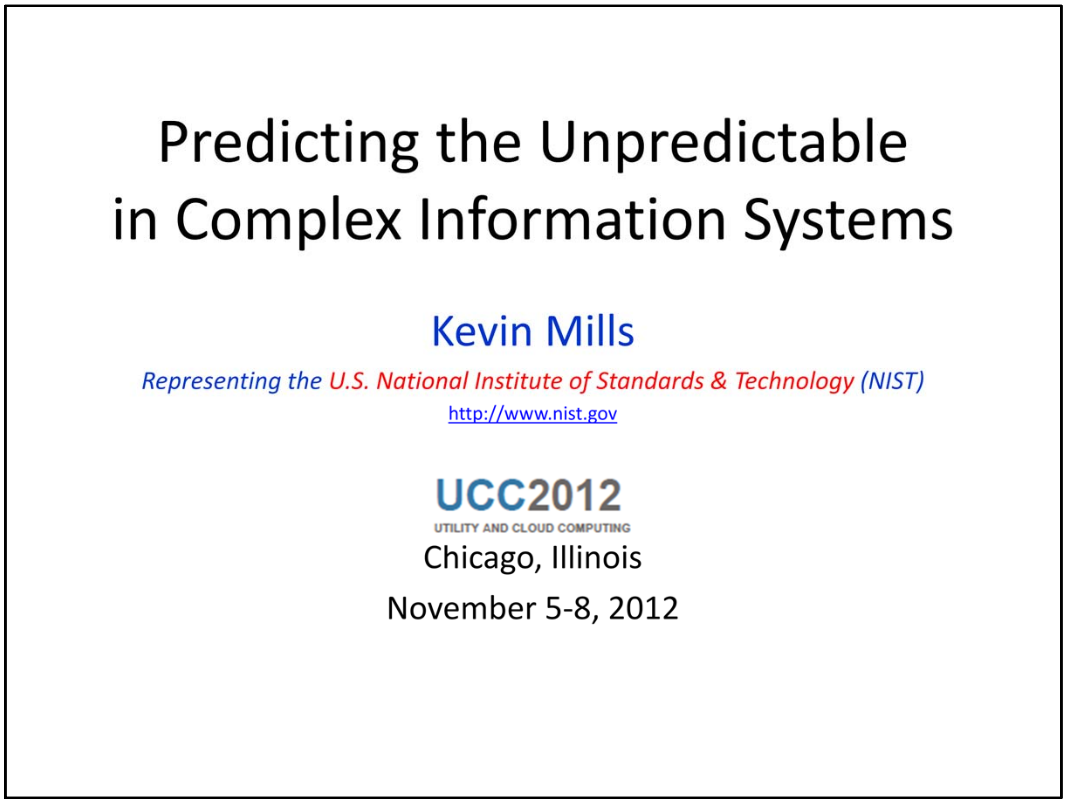


Predicting the Unpredictable in Complex Information Systems

Kevin Mills

Representing the U.S. National Institute of Standards & Technology (NIST)

http://www.nist.gov

UCC2012

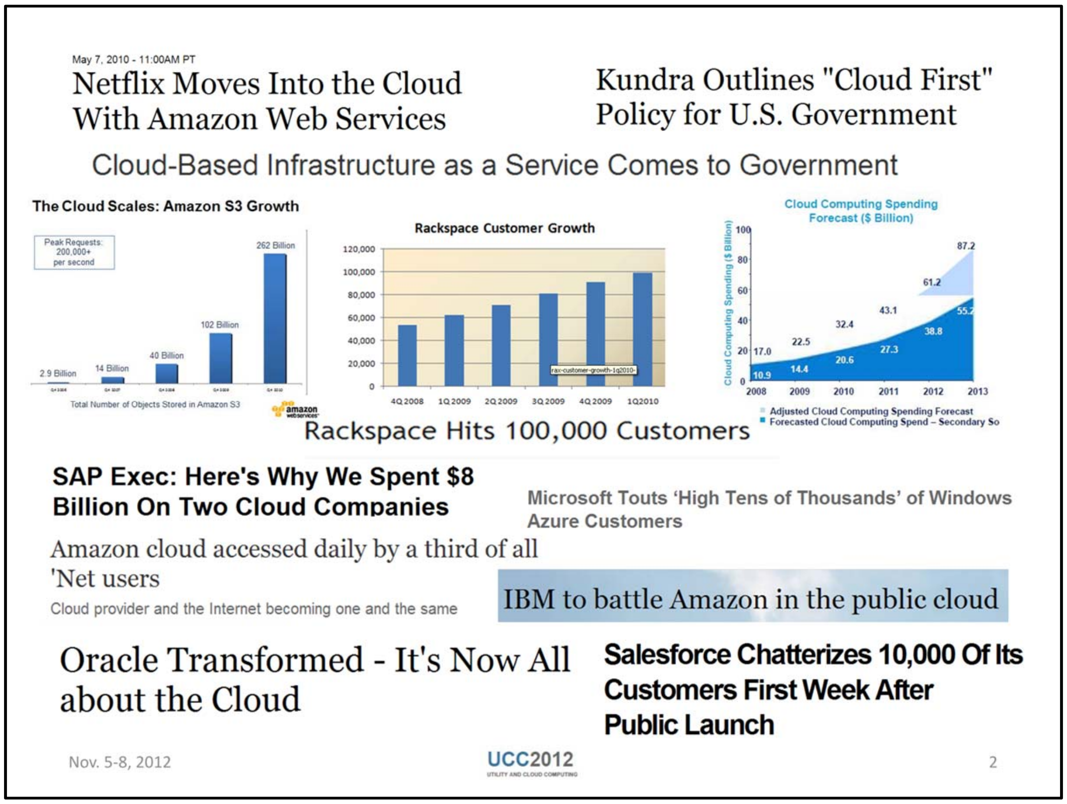
UTILITY AND CLOUD COMPUTING

Chicago, Illinois

November 5-8, 2012

The National Institute of Standards & Technology (NIST) is a measurement laboratory within the U.S. Government. Founded in 1901, NIST is a non-regulatory federal agency within the Department of Commerce. NIST's mission is to promote U.S. innovation and industrial competitiveness by advancing measurement science, standards, and technology in ways that enhance economic security and improve quality of life. NIST is a multidisciplinary organization operating across many scientific domains, including: physics, chemistry, materials and information technology. I work within NIST's Information Technology Laboratory, which also directs the well-known NIST Cloud Computing program. I base much of my talk on computational clouds, though the research applies more generally to complex information systems of all kinds.

To illustrate my thesis, throughout this talk I identify a number of companies offering cloud computing services. While those references encompass a broad cross-section of the industry, I do not identify every cloud-computing company in the market. The fact that I identify certain cloud-computing companies does not imply endorsement by the U.S. Government or NIST, nor does such identification suggest that those companies are the best available.

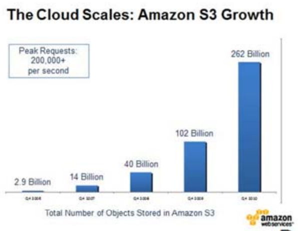


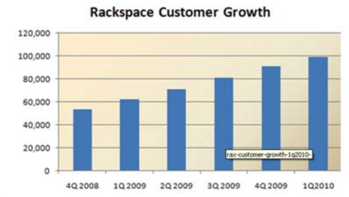
May 7, 2010 - 11:00AM PT

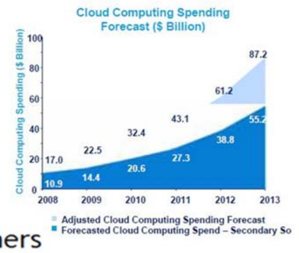
Netflix Moves Into the Cloud
With Amazon Web Services

Kundra Outlines "Cloud First"
Policy for U.S. Government

Cloud-Based Infrastructure as a Service Comes to Government







Rackspace Hits 100,000 Customers

SAP Exec: Here's Why We Spent $8 Billion On Two Cloud Companies

Microsoft Touts 'High Tens of Thousands' of Windows Azure Customers

Amazon cloud accessed daily by a third of all 'Net users

Cloud provider and the Internet becoming one and the same

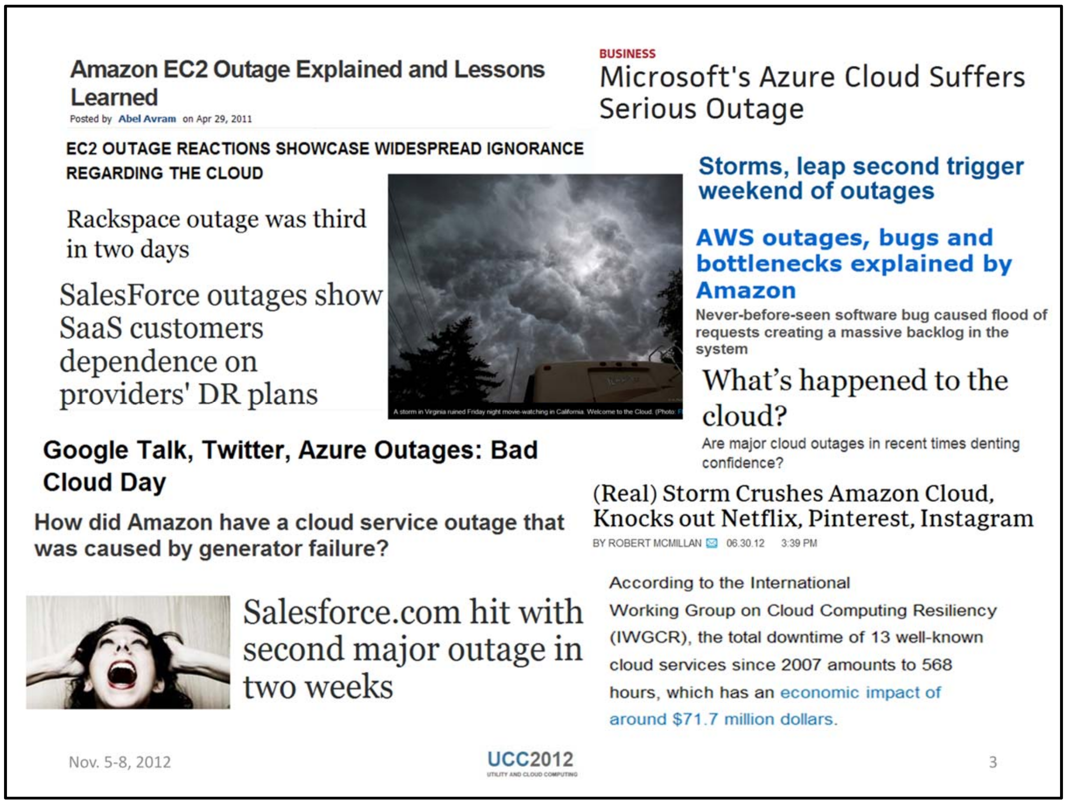
IBM to battle Amazon in the public cloud

Oracle Transformed - It's Now All about the Cloud

Salesforce Chatterizes 10,000 Of Its Customers First Week After Public Launch

Nov. 5-8, 2012 UCC2012 UTILITY AND CLOUD COMPUTING 2

There is an undeniable trend away from corporate-owned data centers and toward adoption of cloud-computing services. The trend appears no matter how it is measured: (1) number of objects stored in the cloud, (2) number of cloud customers or (3) dollars spent on cloud computing. This trend means that our information-based economy exhibits growing dependence on large data centers deployed and managed by a handful of information technology companies. This also implies that enterprises move from paying capital + operating expenses to just paying operating expenses. On the other hand, failures within a handful of large cloud-computing infrastructures can have widespread effects, when compared with failures distributed across a large number of corporate data centers.



Amazon EC2 Outage Explained and Lessons Learned

Posted by Abel Avram on Apr 29, 2011

BUSINESS

Microsoft's Azure Cloud Suffers Serious Outage

EC2 OUTAGE REACTIONS SHOWCASE WIDESPREAD IGNORANCE REGARDING THE CLOUD

Storms, leap second trigger weekend of outages

Rackspace outage was third in two days

AWS outages, bugs and bottlenecks explained by Amazon

Never-before-seen software bug caused flood of requests creating a massive backlog in the system

SalesForce outages show SaaS customers dependence on providers' DR plans

A storm in Virginia ruined Friday night movie-watching in California. Welcome to the Cloud. (Photo:

What's happened to the cloud?

Are major cloud outages in recent times denting confidence?

Google Talk, Twitter, Azure Outages: Bad Cloud Day

(Real) Storm Crushes Amazon Cloud, Knocks out Netflix, Pinterest, Instagram

BY ROBERT MCMILLAN 06.30.12 3:39 PM

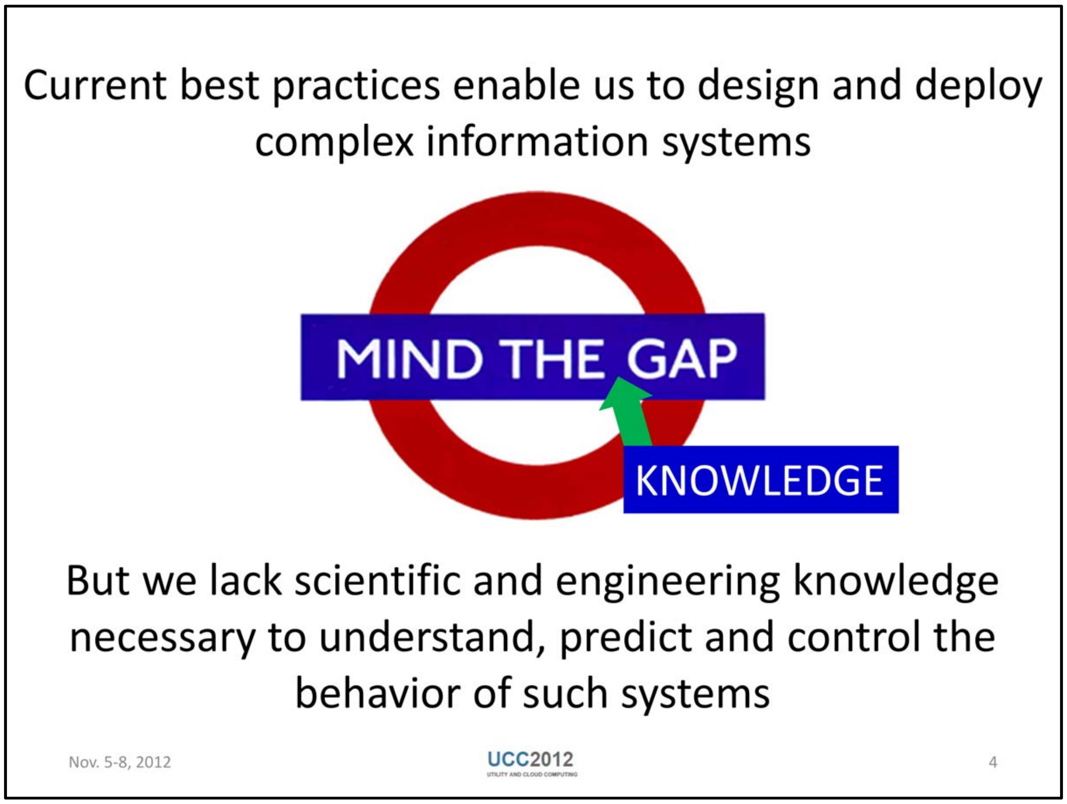
How did Amazon have a cloud service outage that was caused by generator failure?

Salesforce.com hit with second major outage in two weeks

According to the International Working Group on Cloud Computing Resiliency (IWGCR), the total downtime of 13 well-known cloud services since 2007 amounts to 568 hours, which has an economic impact of around $71.7 million dollars.

Nov. 5-8, 2012 UCC2012 UTILITY AND CLOUD COMPUTING 3

Over the past five years, many cloud outages and performance degradations have occurred, and the consequences have been costly. These cloud failures and degradations have been spread widely across the cloud industry rather then being concentrated in one or two cloud vendors. These cloud failures and degradations stem from various root causes, including: power outages due to storms, failures in backup generators, software bugs, configuration errors and insufficient reliability in applications designed for deployment on clouds. Assuming that the rate of such problems stays the same over the coming decade, the growing reliance on cloud services will increase the consequences of failures and degradations, substantially increasing the cost to our information-based economy. Bear in mind that the companies building and using cloud systems and services are quite technically advanced, and very competent in designing and deploying large, distributed systems. Despite this fact, reliability issues continue to plague large, distributed, complex information systems.



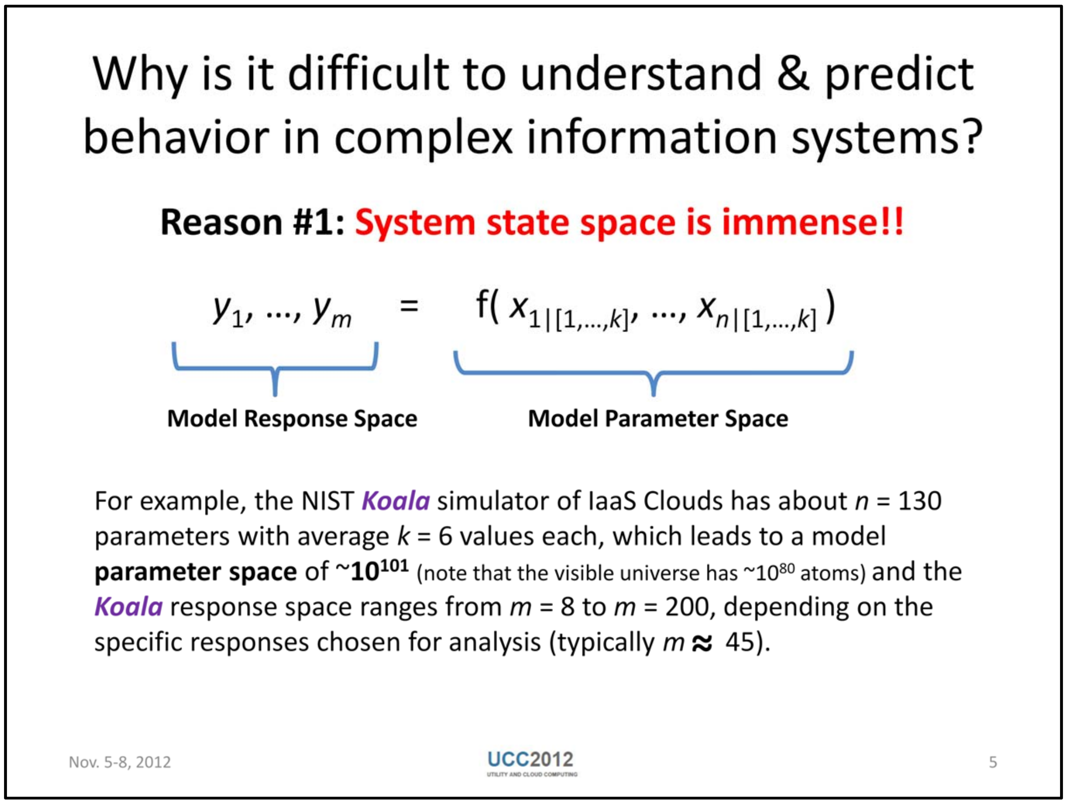
Current best practices enable us to design and deploy complex information systems

But we lack scientific and engineering knowledge necessary to understand, predict and control the behavior of such systems

Nov. 5-8, 2012 UCC2012 UTILITY AND CLOUD COMPUTING 4

My thesis is that failures and performance degradations in such systems do not arise from any inability to design and deploy large-scale systems and distributed applications, but rather from a **knowledge gap**. We lack the scientific and engineering knowledge needed to understand, predict and control behavior in complex information systems, such as clouds, the Internet and distributed systems deployed on them. This knowledge gap threatens to undermine society's growing reliance on the complex information systems that underlie our modern economy.

While there are many reasons why complex information systems are difficult to understand & predict, I am going to focus on three: (1) state-space explosion, (2) emergent behaviors and (3) heavy-tailed distributions.



Why is it difficult to understand & predict behavior in complex information systems?

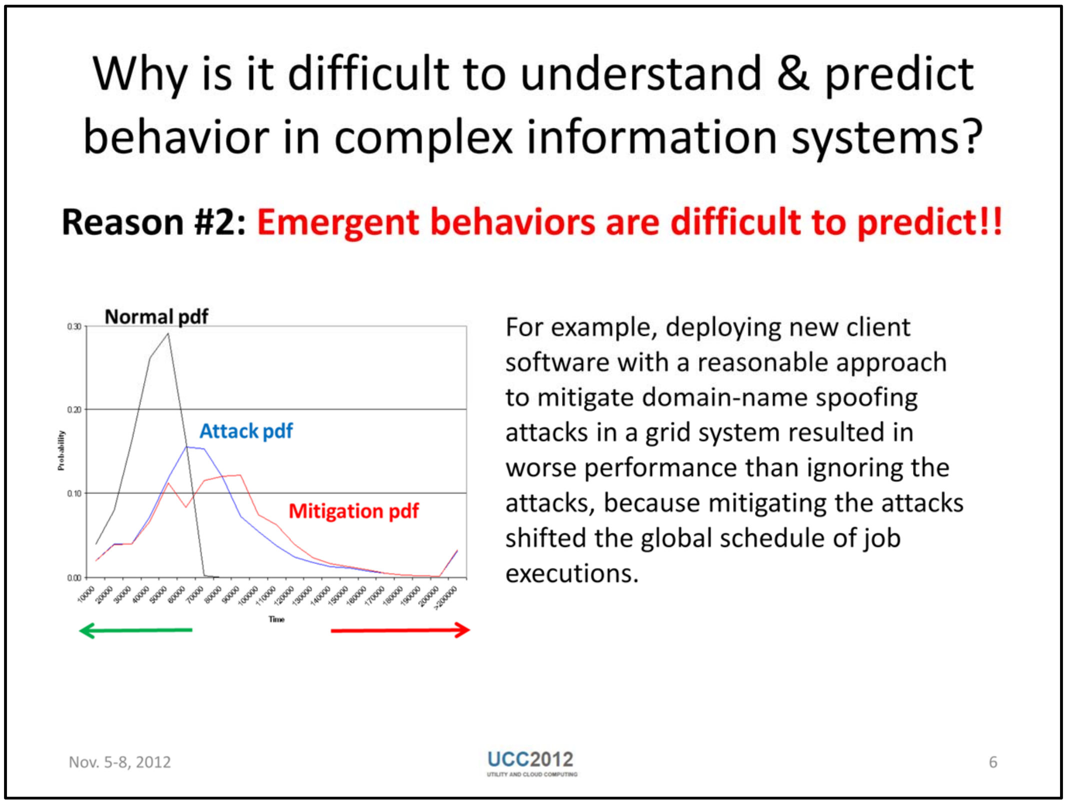
Reason #1: System state space is immense!!

$$\underbrace{y_1, \ldots, y_m}_{\textbf{Model Response Space}} = \underbrace{f(\, x_{1|[1,\ldots,k]}, \ldots, x_{n|[1,\ldots,k]}\,)}_{\textbf{Model Parameter Space}}$$

For example, the NIST ***Koala*** simulator of IaaS Clouds has about $n = 130$ parameters with average $k = 6$ values each, which leads to a model **parameter space** of ~$\mathbf{10^{101}}$ (note that the visible universe has ~10^{80} atoms) and the ***Koala*** response space ranges from $m = 8$ to $m = 200$, depending on the specific responses chosen for analysis (typically $m \approx 45$).

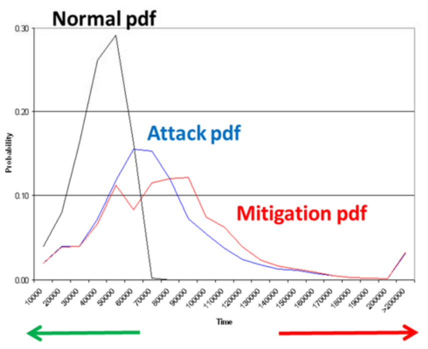
Nov. 5-8, 2012 UCC2012 UTILITY AND CLOUD COMPUTING 5

Complex information systems comprise a vast state space. This means that there are many system states that could exist, and finding those that would lead to system-wide failures and performance degradations is quite challenging. Even in an abstract simulation model of a complex information system the state space is immense. For example, NIST has developed ***Koala***, a simulator of an infrastructure-as-a-service (IaaS) cloud. Koala has about 130 parameters that one might wish to explore for failure scenarios and performance degradations. Though each parameter could be represented by a floating-point number, we can reduce the state-space by restricting the range of parameter values to search to a small subset, say 6 values per parameter (6^{130}). Even with such drastic restriction, the Koala search space encompasses 10^{101} combinations, which exceeds the number of atoms (10^{80}) in the visible universe. Further, simulations can produce myriad responses, some of which are duplicative, others unique and essential. For example, the NIST Koala simulator can exhibit 200 or more responses, which might in reality reflect around a dozen unique behavioral dimensions. When searching the Koala state space, analysts must be sure to measure all the unique behaviors and not to overweigh overlapping responses.



Why is it difficult to understand & predict behavior in complex information systems?

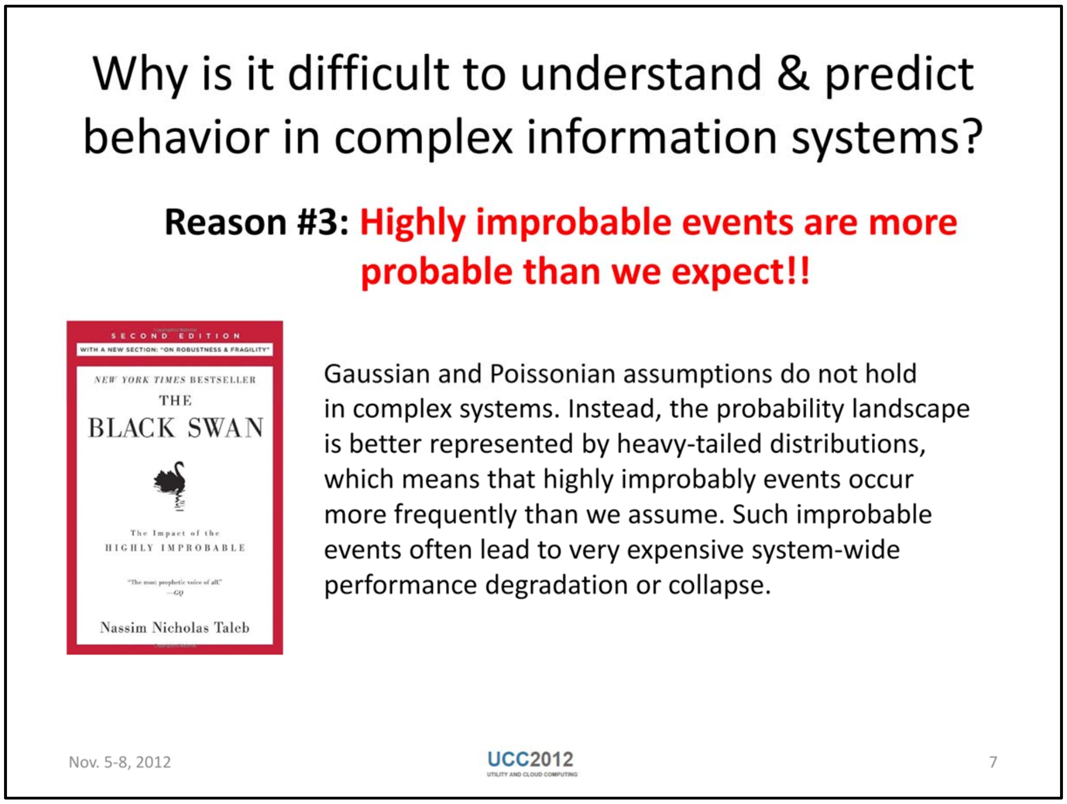
Reason #2: Emergent behaviors are difficult to predict!!



For example, deploying new client software with a reasonable approach to mitigate domain-name spoofing attacks in a grid system resulted in worse performance than ignoring the attacks, because mitigating the attacks shifted the global schedule of job executions.

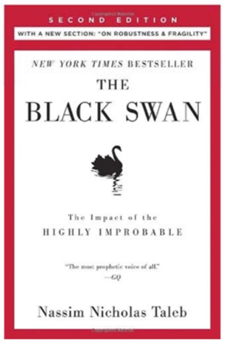
Nov. 5-8, 2012 UCC2012 UTILITY AND CLOUD COMPUTING 6

Many key, global properties of complex information systems appear as emergent behaviors, which can be impossible to predict based on detailed analyses of the behavior of individual system components. Such global properties can only be discovered by measuring macroscopic variables of a system under an appropriate set of conditions that stimulate the spatiotemporal emergence of the behaviors of interest. Changing system components in some way intended to create a particular outcome can result in global emergent behavior that is quite different than expected. For example, we simulated a distributed grid computing system, where a population of users attempted to complete multitask, workflow jobs. The graph plots (y axis) the probability density function (pdf) of the times (x axis) taken to complete jobs. The black curve represents baseline system performance. Subsequently, we injected a domain-name system (DNS) spoofing attack, where some sites pretended to be willing to execute user tasks, accepted user submissions and then did execute the jobs. Affected users had to detect such events and then resubmit their jobs to other, real service providers. As a result, the blue curve shows the effects of DNS spoofing on the pdf of job execution times. We then imagined the developer of the grid client released a software update that contained sensible behavior to combat DNS spoofing. Specifically, once an suspected spoofer is detected, the client places the suspect in a penalty box and does not attempt to use the suspect until some penalty period expires. As the red curve shows, the updated client software worsened rather than improved system performance, causing a larger proportion of jobs to be later than before the patch was released. This result occurred because the global schedule of task executions is an emergence property of a grid system. By improving the ability of clients to combat DNS spoofing attacks, more clients became competitive in the same time period for the limited processing resources in the grid, and the schedule of task executions was shifted, causing more work-flows to be delayed in their overall completion times.



Why is it difficult to understand & predict behavior in complex information systems?

Reason #3: Highly improbable events are more probable than we expect!!

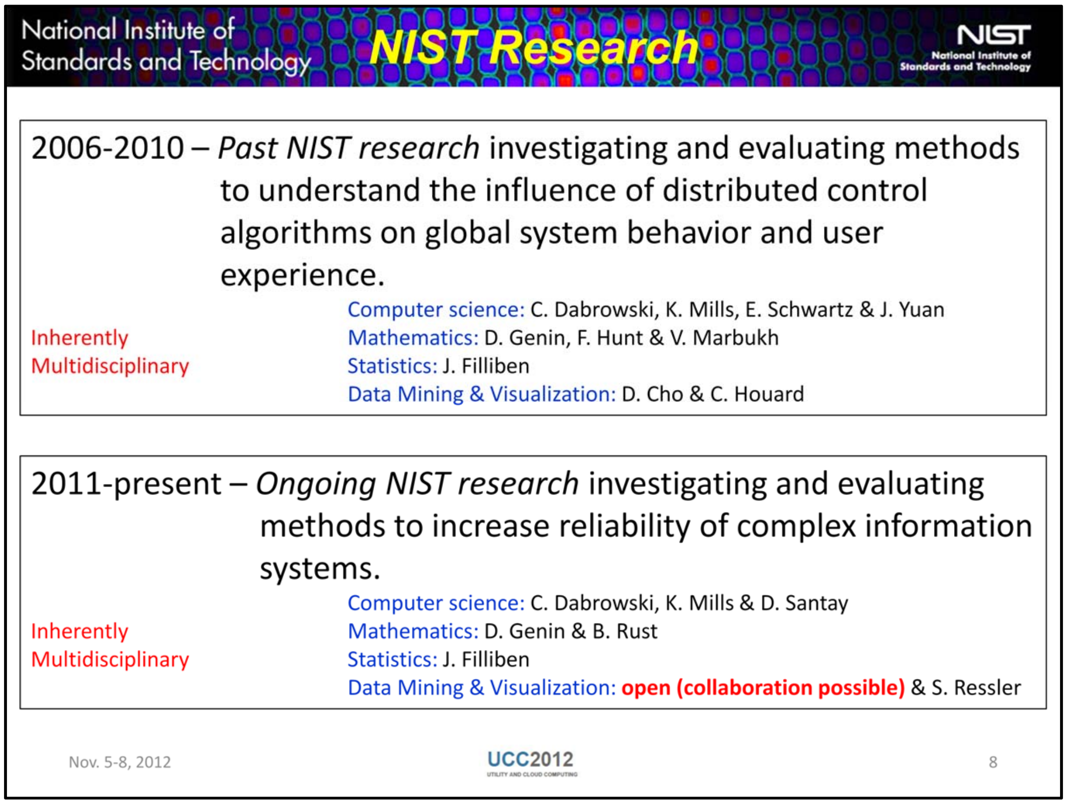

SECOND EDITION
WITH A NEW SECTION: "ON ROBUSTNESS & FRAGILITY"
NEW YORK TIMES BESTSELLER
THE BLACK SWAN
The Impact of the HIGHLY IMPROBABLE
Nassim Nicholas Taleb

Gaussian and Poissonian assumptions do not hold in complex systems. Instead, the probability landscape is better represented by heavy-tailed distributions, which means that highly improbably events occur more frequently than we assume. Such improbable events often lead to very expensive system-wide performance degradation or collapse.

Nov. 5-8, 2012 UCC2012 UTILITY AND CLOUD COMPUTING 7

In a best-selling book, Nassim Nicholas Taleb dubbed rare events as black swans, which appear infrequently in a world where swans are mainly white. In complex information systems, large-scale system failures can often be attributable to such rare events, which can lead to very costly outcomes. Most traditional analytical modeling of information systems adopts Gaussian or Poissonian assumptions because the underlying probability distributions are amenable to tractable mathematical analyses. In an effort to validate such analytical models, simulations of information systems usually adopt the same assumptions. In such probability distributions, the likelihood of improbable events, > 3 sigma from the mean, is vanishingly small, thus rare, potentially costly, outcomes occur quite infrequently. Taleb argues that Gaussian assumptions do not adequately represent the true probability distribution of rare events in complex systems. Instead, Taleb proposes that heavy-tailed distributions, such as Pareto and log-normal more closely represent reality. In such heavy-tailed distributions, the occurrence of rare events is more highly probable than in a Gaussian distribution. In other words, rare events that can lead to system-wide failures are more likely then assumed in the usual approach to systems modeling. By adopting optimistic assumptions when simulating system behavior, analysts obtain a biased and erroneous view about the potential for unexpected rare events to cause disruptions.

For the past six years, my colleagues and I at NIST have been investigating techniques that can be used to understand & predict behavior in complex information systems. Next I will give a high-level overview of our research. Then, I will focus on one particular research objective that I have been pursuing over the past year.


National Institute of Standards and Technology

NIST Research

NIST National Institute of Standards and Technology

2006-2010 – *Past NIST research* investigating and evaluating methods to understand the influence of distributed control algorithms on global system behavior and user experience.

Inherently Multidisciplinary

Computer science: C. Dabrowski, K. Mills, E. Schwartz & J. Yuan
Mathematics: D. Genin, F. Hunt & V. Marbukh
Statistics: J. Filliben
Data Mining & Visualization: D. Cho & C. Houard

2011-present – *Ongoing NIST research* investigating and evaluating methods to increase reliability of complex information systems.

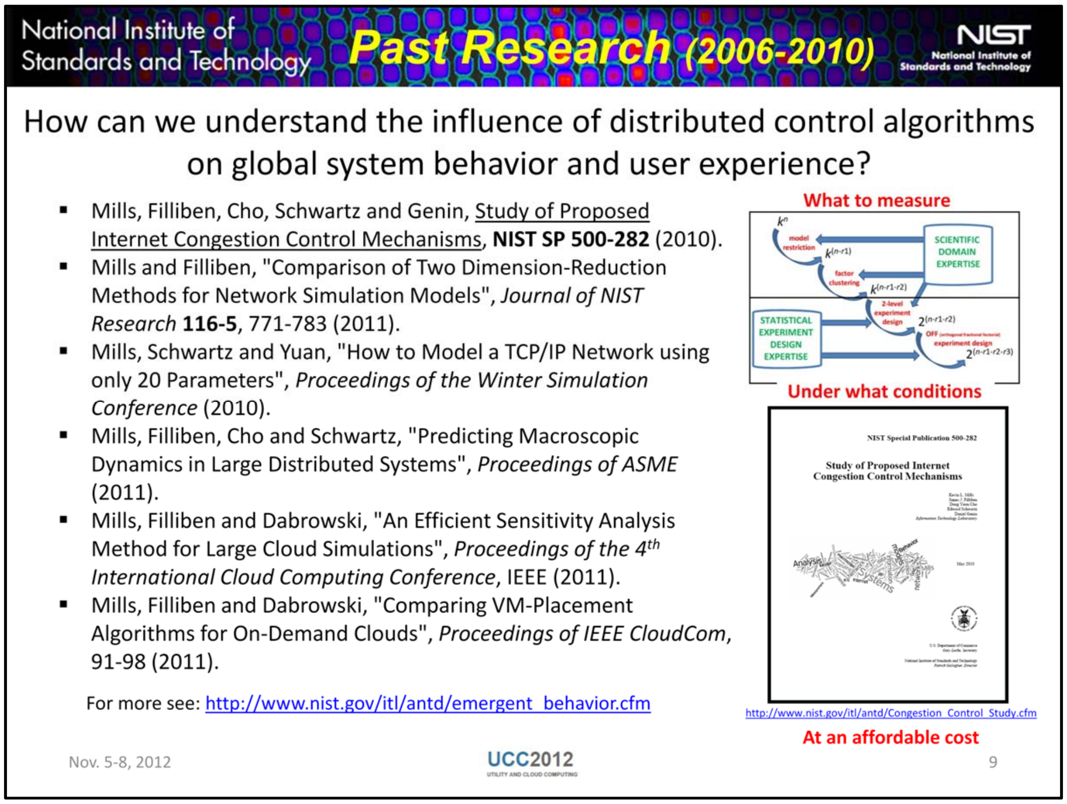
Inherently Multidisciplinary

Computer science: C. Dabrowski, K. Mills & D. Santay
Mathematics: D. Genin & B. Rust
Statistics: J. Filliben
Data Mining & Visualization: **open (collaboration possible)** & S. Ressler

Nov. 5-8, 2012 UCC2012 UTILITY AND CLOUD COMPUTING 8

From 2006-2010, my colleagues & I investigated and evaluated methods to understand the influence of distributed control algorithms on global system behavior and on user experience. We focused on two case studies: (1) comparing proposed replacements for the TCP congestion-control algorithm and (2) comparing alternative algorithms for allocating virtual machines to physical machines in infrastructure clouds. The work was general and inherently multidisciplinary, spanning computer science, mathematics, statistics, data mining and information visualization. As you will see, this work was quite successful.

Beginning in 2011, we turned our attention to investigating and evaluating methods to increase reliability in complex information systems. Our focus to date has been on infrastructure clouds, where we could leverage the Koala simulator developed under our previous research. The work remains inherently multidisciplinary, though at the present time we have an opening for data-mining expertise, because one of my colleagues, Dong Yeon Cho, left NIST for NIH, where he now works on data mining for biological research. Dr. Cho's data-mining expertise proved invaluable, which means we have a need, but also means we have a possibility to collaborate with someone who has interest and expertise in data mining. If you are interested please let me know, either at the conference or later via email.



National Institute of Standards and Technology

Past Research (2006-2010)

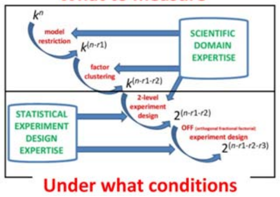
NIST National Institute of Standards and Technology

How can we understand the influence of distributed control algorithms on global system behavior and user experience?

- Mills, Filliben, Cho, Schwartz and Genin, Study of Proposed Internet Congestion Control Mechanisms, **NIST SP 500-282** (2010).
- Mills and Filliben, "Comparison of Two Dimension-Reduction Methods for Network Simulation Models", *Journal of NIST Research* **116-5**, 771-783 (2011).
- Mills, Schwartz and Yuan, "How to Model a TCP/IP Network using only 20 Parameters", *Proceedings of the Winter Simulation Conference* (2010).
- Mills, Filliben, Cho and Schwartz, "Predicting Macroscopic Dynamics in Large Distributed Systems", *Proceedings of ASME* (2011).
- Mills, Filliben and Dabrowski, "An Efficient Sensitivity Analysis Method for Large Cloud Simulations", *Proceedings of the 4th International Cloud Computing Conference*, IEEE (2011).
- Mills, Filliben and Dabrowski, "Comparing VM-Placement Algorithms for On-Demand Clouds", *Proceedings of IEEE CloudCom*, 91-98 (2011).

For more see: http://www.nist.gov/itl/antd/emergent_behavior.cfm

What to measure



k^n
model restriction
$k^{(n-r1)}$
SCIENTIFIC DOMAIN EXPERTISE
factor clustering
$k^{(n-r1-r2)}$
2-level experiment design
STATISTICAL EXPERIMENT DESIGN EXPERTISE
$2^{(n-r1-r2)}$
OFF (orthogonal fractional factorial) experiment design
$2^{(n-r1-r2-r3)}$

Under what conditions

NIST Special Publication 500-282

Study of Proposed Internet Congestion Control Mechanisms

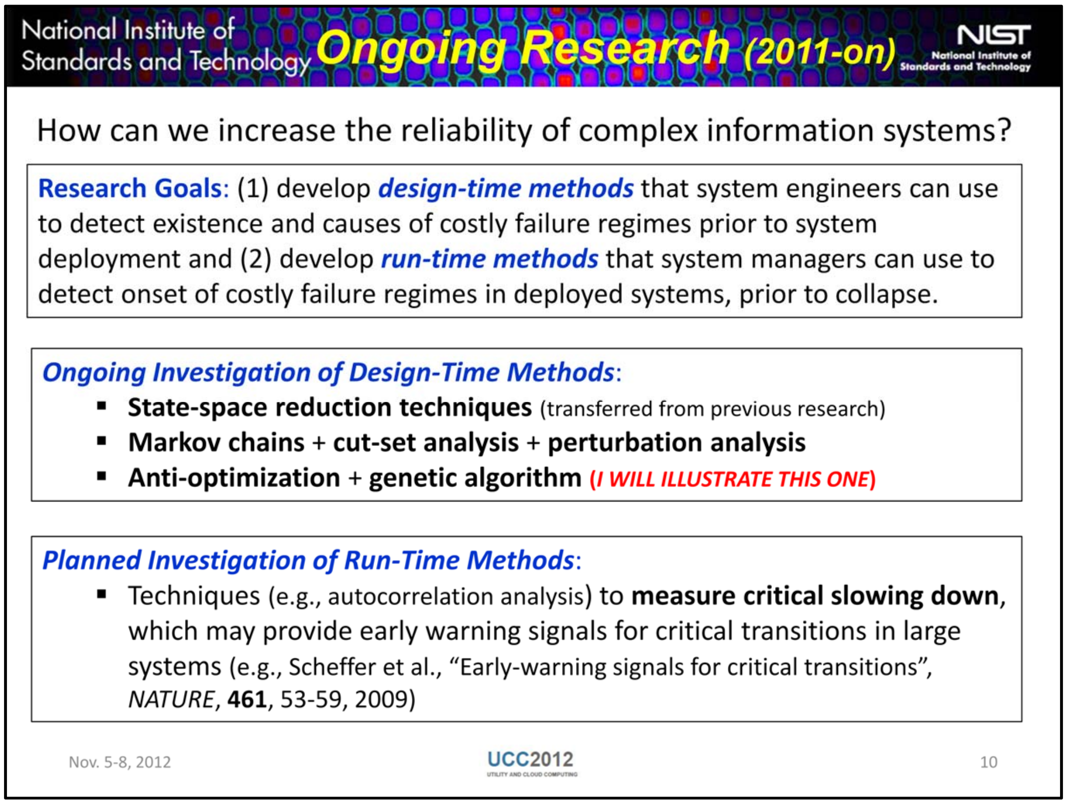
http://www.nist.gov/itl/antd/Congestion_Control_Study.cfm

At an affordable cost

Nov. 5-8, 2012 UCC2012 UTILITY AND CLOUD COMPUTING 9

The primary focus of our past research was on: (1) state-space reduction methods (for parameters and responses) and (2) measuring global system behaviors. Regarding state-space reduction, we employed a combination of model restriction, factor clustering and 2-level orthogonal fractional factorial experiment design to reduce the experiment parameter space for a TCP/IP network model from 10^{539} to 2^5. Using correlation analysis and clustering, we reduced the response space of the model from 22 responses to only 7 unique behaviors.
These reductions enabled us to compare TCP with 7 proposed replacements under only 32 conditions, requiring 256 simulations for a given experiment scenario. Such parsimony allowed us to investigate 5 scenarios, with (8 x 32 x 5 =) 1280 simulations, leading to the most comprehensive study to date of proposed TCP replacements. As our report shows: (1) we found that the proposed replacements would not significantly change global network behavior, and we were able to explain why, (2) we identified 5 conditions that must hold for a user to gain advantage from any of the proposed TCP replacements, (3) we identified one of the proposed TCP replacements as most suitable for deployment and (4) we raised cautionary notes about another proposed TCP replacement. To demonstrate generality, we applied the same methods to compare alternative VM-placement algorithms for on-demand clouds. In summary, our research from 2006 to 2010 focused mainly on methods to determine what to measure under what conditions at an affordable cost when comparing competing distributed control algorithms for complex information systems.

While our state-space reduction methods allow us to characterize global system behaviors under a wide range of parameter combinations, these approaches do not address directly the search for low-probability, highly costly failure scenarios and performance degradations. That is the focus of our ongoing research.


National Institute of Standards and Technology

Ongoing Research (2011-on)

NIST National Institute of Standards and Technology

How can we increase the reliability of complex information systems?

Research Goals: (1) develop ***design-time methods*** that system engineers can use to detect existence and causes of costly failure regimes prior to system deployment and (2) develop ***run-time methods*** that system managers can use to detect onset of costly failure regimes in deployed systems, prior to collapse.

Ongoing Investigation of Design-Time Methods:

- **State-space reduction techniques** (transferred from previous research)
- **Markov chains** + **cut-set analysis** + **perturbation analysis**
- **Anti-optimization** + **genetic algorithm** (*I WILL ILLUSTRATE THIS ONE*)

Planned Investigation of Run-Time Methods:

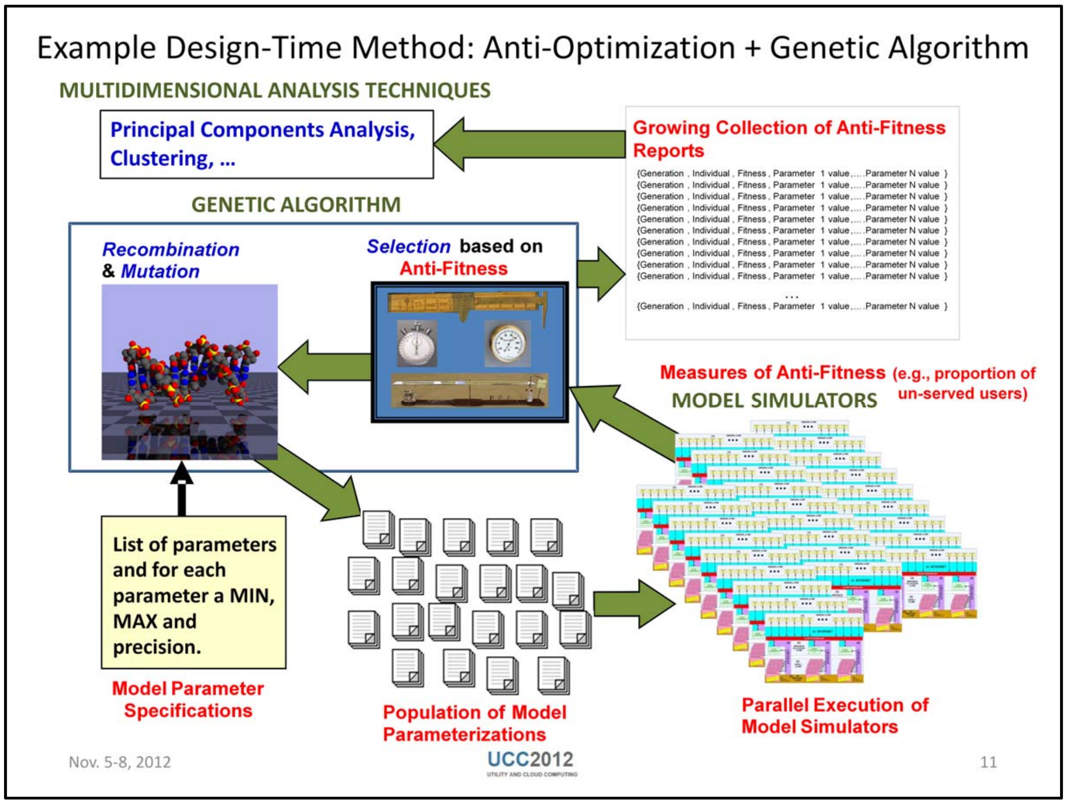
- Techniques (e.g., autocorrelation analysis) to **measure critical slowing down**, which may provide early warning signals for critical transitions in large systems (e.g., Scheffer et al., “Early-warning signals for critical transitions”, *NATURE*, **461**, 53-59, 2009)

Nov. 5-8, 2012 UCC2012 UTILITY AND CLOUD COMPUTING 10

To increase reliability in complex information systems, we are investigating two general classes of methods: (1) design-time methods and (2) run-time methods. We aim for design-time methods that system engineers can use to detect existence and causes of costly failure regimes prior to system deployment. In that way, black swans can be identified and planned for. Since all design-time failure scenarios cannot be uncovered a priori, we also seek run-time methods that system operators can use to detect onset of costly failure scenarios in deployed systems, prior to system collapse.

We are investigating three design-time methods. First, we hope to exploit state-space reduction techniques from our previous research to narrow the parameter combinations that need to be considered. Second, we are constructing Markov chain models, with probabilities determined by instrumenting systems or models operating under nominal conditions, and then treating those models as graphs, which can be subjected to cut-set analysis to determine subsets of edges that would disconnect the graph if removed. These subsets represent potential failure scenarios. We then use perturbation analysis on these subsets to determine numeric thresholds at which the system fails, and also collapse trajectories. We have published preliminary work on this approach. We are also investigating the use of genetic algorithms to drive system models into anti-optimal behavioral directions, I will focus on this throughout the rest of my talk.

For run-time methods, we are motivated by a *Nature* article, where an examination of natural systems, fro cell signaling pathways to ecosystems and the climate, found that such complex systems exhibit a critical slowing down in response to perturbations, as those systems near a critical point between two behavioral regimes. We are investigating whether complex information systems exhibit similar traits, and whether various signal-analysis techniques could identify critical slowing down in such systems. Let me turn now to the use of genetic algorithms in design-time methods.



Example Design-Time Method: Anti-Optimization + Genetic Algorithm

MULTIDIMENSIONAL ANALYSIS TECHNIQUES

Principal Components Analysis, Clustering, …

Growing Collection of Anti-Fitness Reports

{Generation , Individual , Fitness , Parameter 1 value,....Parameter N value }
{Generation , Individual , Fitness , Parameter 1 value,....Parameter N value }
{Generation , Individual , Fitness , Parameter 1 value,....Parameter N value }
{Generation , Individual , Fitness , Parameter 1 value,....Parameter N value }
{Generation , Individual , Fitness , Parameter 1 value,....Parameter N value }
{Generation , Individual , Fitness , Parameter 1 value,....Parameter N value }
{Generation , Individual , Fitness , Parameter 1 value,....Parameter N value }
{Generation , Individual , Fitness , Parameter 1 value,....Parameter N value }
{Generation , Individual , Fitness , Parameter 1 value,....Parameter N value }
{Generation , Individual , Fitness , Parameter 1 value,....Parameter N value }
…
{Generation , Individual , Fitness , Parameter 1 value,....Parameter N value }

GENETIC ALGORITHM

Recombination & *Mutation*

Selection based on Anti-Fitness

Measures of Anti-Fitness (e.g., proportion of un-served users)

MODEL SIMULATORS

List of parameters and for each parameter a MIN, MAX and precision.

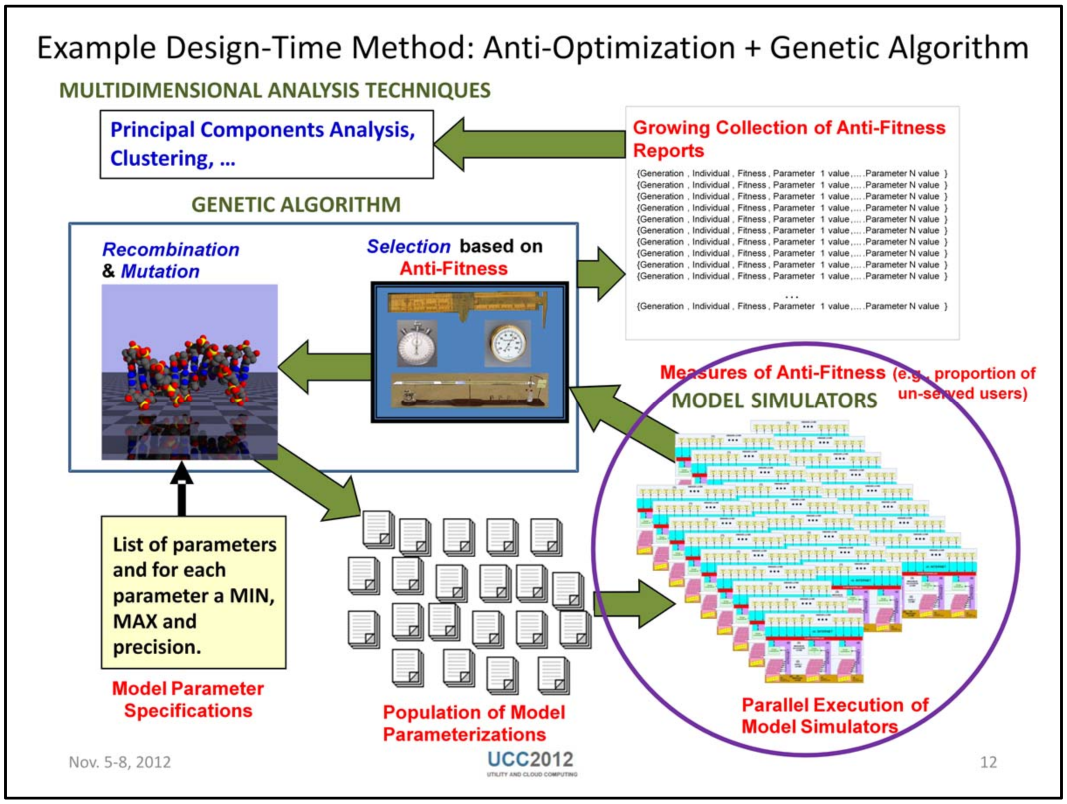
Model Parameter Specifications

Population of Model Parameterizations

Parallel Execution of Model Simulators

Nov. 5-8, 2012 UCC2012 UTILITY AND CLOUD COMPUTING 11

Historically, genetic algorithms (GAs) have been used to search for optimal solutions to various, challenging problems. In our application, we invert the use of GAs so that they search for anti-optimal solutions, which may indicate system failure scenarios or performance degradation. To accomplish this, we define some measure of anti-fitness, e.g., the proportion of un-served users, and then deploy a GA to search for model parameter combinations that maximize anti-fitness. Classic GAs encode parameters as bit strings, which can be manipulated through various operations, e.g., selection, recombination and mutation, mimicking genetic reproduction, which, when carried out in populations over many generations, also mimics evolution. For our application, we identify a set of parameters to search, and then define the bounds and granularity of each parameter, which can then be represented as bit strings. We write some code that transforms bit strings into simulator parameter files. The GA begins by generating a random set of parameters for a parallel population of simulators. The simulators then execute until finished (or time expires) and report their anti-fitness measures. The GA uses these measures to select a candidate set of parameters on which to base the next generation. The GA then applies crossover-recombination operators to pairs of the selected candidates, followed by probabilistic mutation of the recombined pairs. The resulting bit strings are used to re-parameterize the population of simulators, which again execute until finished (or time expires). This continues until a preset number of generations has been completed. At the end of each generation, the GA records a collection of tuples that each report (one per simulation) the parameter settings and anti-fitness achieved. Off-line, using various data mining techniques, the tuple collection is analyzed to create clusters of parameter combinations that could be interpreted to represent categories of reasons why a system might fail or degrade. Next, I will discuss four specific aspects of this application of GAs, as illustrated in the context of the Koala cloud simulator.



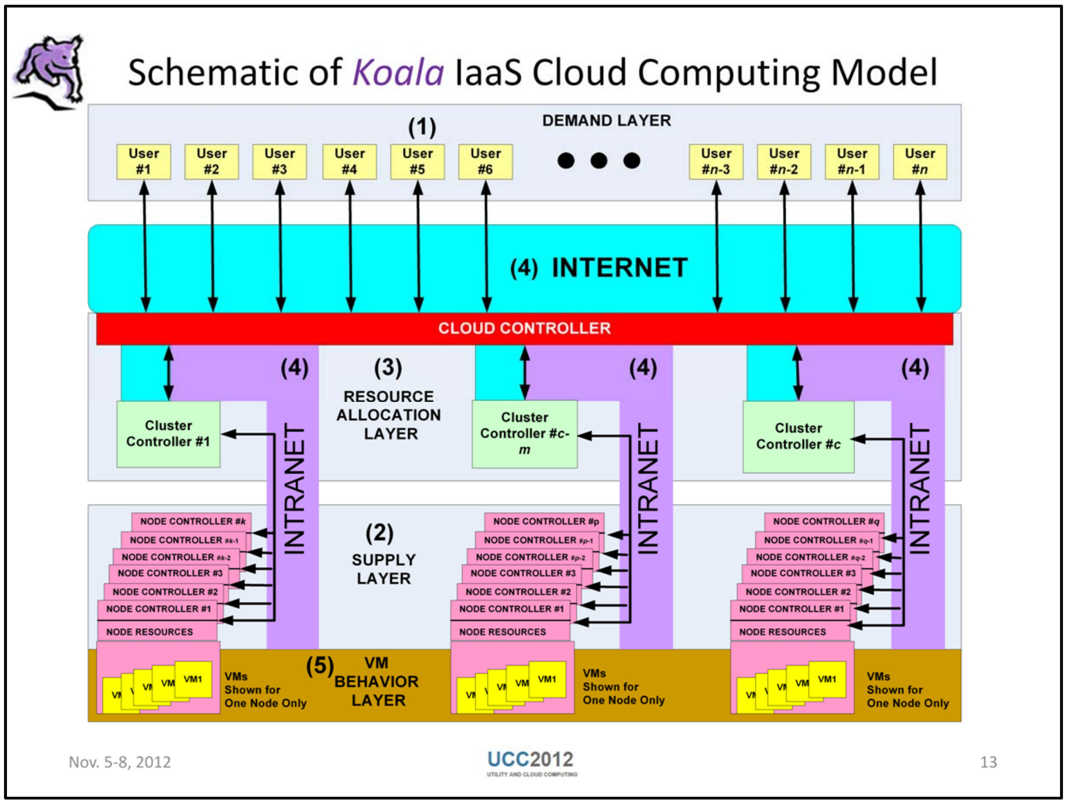
Example Design-Time Method: Anti-Optimization + Genetic Algorithm

MULTIDIMENSIONAL ANALYSIS TECHNIQUES
Principal Components Analysis, Clustering, …
Growing Collection of Anti-Fitness Reports
GENETIC ALGORITHM
Recombination & Mutation
Selection based on Anti-Fitness
Measures of Anti-Fitness (e.g., proportion of un-served users)
MODEL SIMULATORS
List of parameters and for each parameter a MIN, MAX and precision.
Model Parameter Specifications
Population of Model Parameterizations
Parallel Execution of Model Simulators

{Generation , Individual , Fitness , Parameter 1 value,….Parameter N value }
{Generation , Individual , Fitness , Parameter 1 value,….Parameter N value }
{Generation , Individual , Fitness , Parameter 1 value,….Parameter N value }
{Generation , Individual , Fitness , Parameter 1 value,….Parameter N value }
{Generation , Individual , Fitness , Parameter 1 value,….Parameter N value }
{Generation , Individual , Fitness , Parameter 1 value,….Parameter N value }
{Generation , Individual , Fitness , Parameter 1 value,….Parameter N value }
{Generation , Individual , Fitness , Parameter 1 value,….Parameter N value }
{Generation , Individual , Fitness , Parameter 1 value,….Parameter N value }
{Generation , Individual , Fitness , Parameter 1 value,….Parameter N value }
…
{Generation , Individual , Fitness , Parameter 1 value,….Parameter N value }

Nov. 5-8, 2012 UCC2012 UTILITY AND CLOUD COMPUTING 12

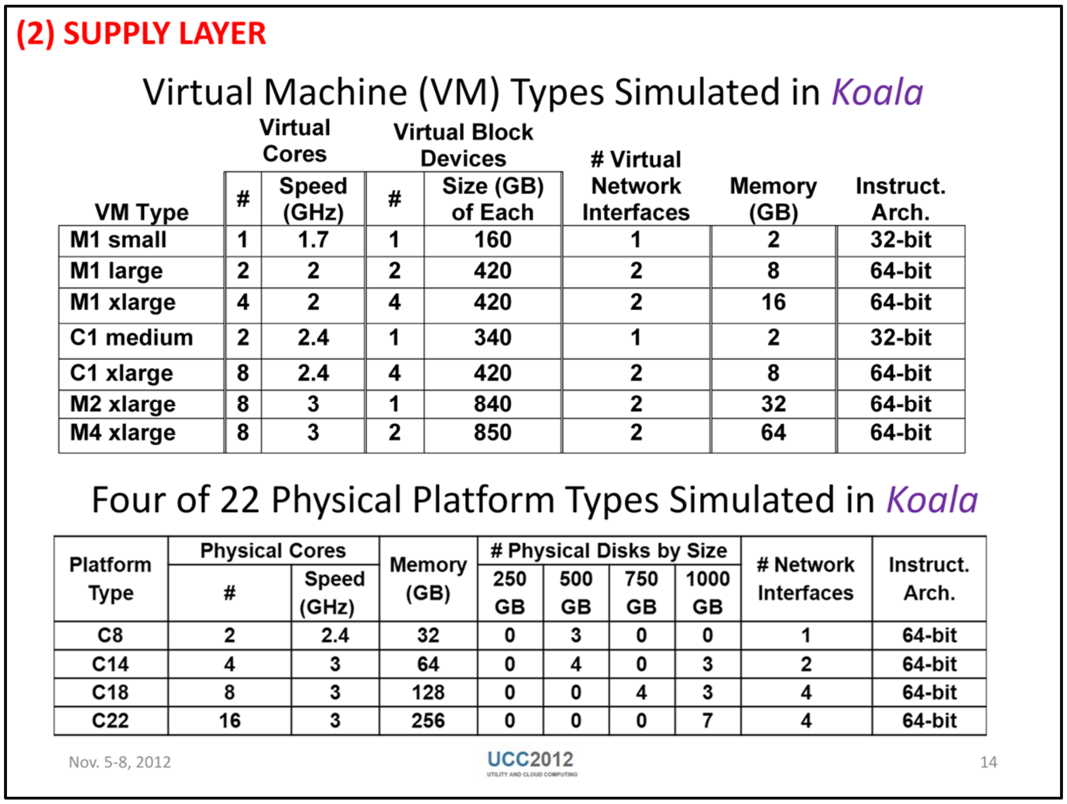
I will begin with an overview of the Koala cloud simulator.



Schematic of *Koala* IaaS Cloud Computing Model

Nov. 5-8, 2012 UCC2012 UTILITY AND CLOUD COMPUTING 13

Koala simulates virtual-machine placement in an infrastructure-as-a-service (IaaS) cloud. The model consists of five layers: (1) demand, (2) supply, (3) resource allocation, (4) internet/intranet and (5) virtual-machine behavior. We use only layers (1)-(4) in the current experiment, which focuses on virtual-machine placement. The simulated cloud offers a set of physical nodes that may be shared among users by placing one or more virtual machines on each node. The simulated cloud offers only a limited set of virtual-machine types. Users can request multiple instances of one or more virtual-machine types. Through the Internet, users contact a cloud controller to request virtual machines. The cloud controller consults subordinate cluster controllers to see if any cluster can accept the user's request. Koala maps all virtual machines for a single user onto the same cluster, in order to benefit from cluster-local communications. The cloud and cluster controllers communicate over either the Internet or an intranet, depending on how the model is configured. Cluster controllers decide on which physical nodes to place each virtual machine that is assigned by the cloud controller. Cluster controllers interact across an intranet with nodes, via a node controller that can boot, reboot, terminate and monitor the execution of virtual machines. The virtual-machine behavior layer allows virtual machines to consume node resources and allows node controllers to monitor resource usage by executing virtual machines. The user-cloud interface is modeled after Amazon EC2, while the internal structure of the cloud is patterned loosely after the public-domain version (1.6) of Eucalyptus.



(2) SUPPLY LAYER

Virtual Machine (VM) Types Simulated in *Koala*

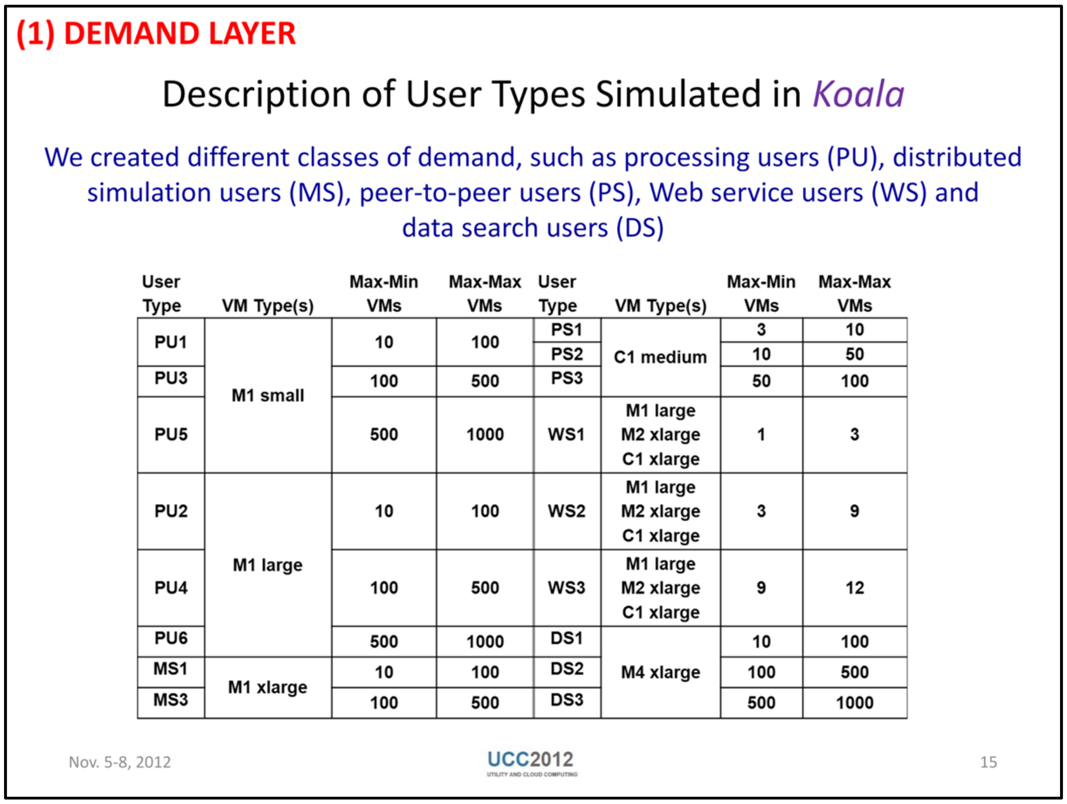
VM Type	Virtual Cores #	Virtual Cores Speed (GHz)	Virtual Block Devices #	Virtual Block Devices Size (GB) of Each	# Virtual Network Interfaces	Memory (GB)	Instruct. Arch.
M1 small	1	1.7	1	160	1	2	32-bit
M1 large	2	2	2	420	2	8	64-bit
M1 xlarge	4	2	4	420	2	16	64-bit
C1 medium	2	2.4	1	340	1	2	32-bit
C1 xlarge	8	2.4	4	420	2	8	64-bit
M2 xlarge	8	3	1	840	2	32	64-bit
M4 xlarge	8	3	2	850	2	64	64-bit

Four of 22 Physical Platform Types Simulated in *Koala*

Platform Type	Physical Cores #	Physical Cores Speed (GHz)	Memory (GB)	# Physical Disks by Size 250 GB	500 GB	750 GB	1000 GB	# Network Interfaces	Instruct. Arch.
C8	2	2.4	32	0	3	0	0	1	64-bit
C14	4	3	64	0	4	0	3	2	64-bit
C18	8	3	128	0	0	4	3	4	64-bit
C22	16	3	256	0	0	0	7	4	64-bit

Nov. 5-8, 2012 UCC2012 UTILITY AND CLOUD COMPUTING 14

The supply layer has two different views. In one view, the cloud offers users a small number (7) of virtual-machine types, which define the number and speed of virtual cores, the number and size of virtual disks, the number of virtual network interfaces, the amount of memory and the instruction-set architecture. In the second view, the supply layer consists of physical nodes that can assume a range of types (22). Here, we show only four of the 22 Koala platform types. Physical nodes are defined by the number and speed of physical cores, the amount of memory, the number and size of physical disks, the number of physical network interfaces and the instruction-set architecture. In general, physical nodes may be much bigger than virtual machines, so multiple virtual machines may share a single physical node. The virtual-machine-placement problem involves allocating virtual machines to physical nodes in such a way as to minimize the amount of idle capacity in physical nodes. The problem can be mapped mathematically to a bin-packing problem.



(1) DEMAND LAYER

Description of User Types Simulated in *Koala*

We created different classes of demand, such as processing users (PU), distributed simulation users (MS), peer-to-peer users (PS), Web service users (WS) and data search users (DS)

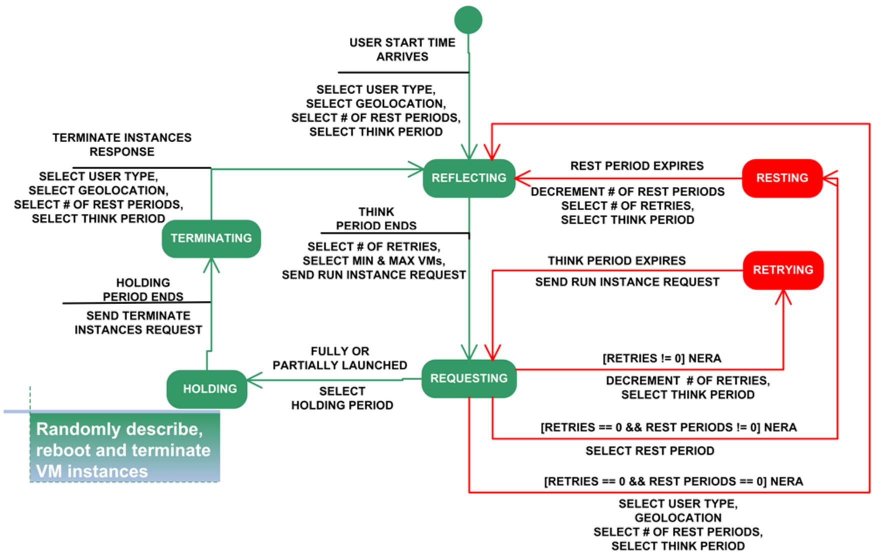
User Type	VM Type(s)	Max-Min VMs	Max-Max VMs	User Type	VM Type(s)	Max-Min VMs	Max-Max VMs
PU1	M1 small	10	100	PS1	C1 medium	3	10
				PS2		10	50
PU3		100	500	PS3		50	100
PU5		500	1000	WS1	M1 large M2 xlarge C1 xlarge	1	3
PU2	M1 large	10	100	WS2	M1 large M2 xlarge C1 xlarge	3	9
PU4		100	500	WS3	M1 large M2 xlarge C1 xlarge	9	12
PU6		500	1000	DS1	M4 xlarge	10	100
MS1	M1 xlarge	10	100	DS2		100	500
MS3		100	500	DS3		500	1000

Nov. 5-8, 2012 UCC2012 UTILITY AND CLOUD COMPUTING 15

Koala allows users to assume any of 21 different types, of which I show 17 here. A user type defines a class of simulated demand, such as processing users, distributed simulation users, P2P users, Web service users and data-search users. Any user type is defined by one of more virtual-machine types that is required. For each virtual-machine type the user requests a minimum number of virtual-machine instances needed to compute a job, along with a maximum number needed. Under one parameterization, users select the minimum number of instances uniformly distributed from 1 to max-min and the maximum number of instances uniformly distributed from max-min to max-max. Under another parameterization, users simply set the minimum to max-min and the maximum to max-max. If a user cannot obtain at least the minimum number requested for each virtual-machine type, then the user receives a NERA, not enough resources available, fault and must retry later.

(1) DEMAND LAYER - USER BEHAVIOR

Finite-State Machine of Simulated User Behavior in *Koala*

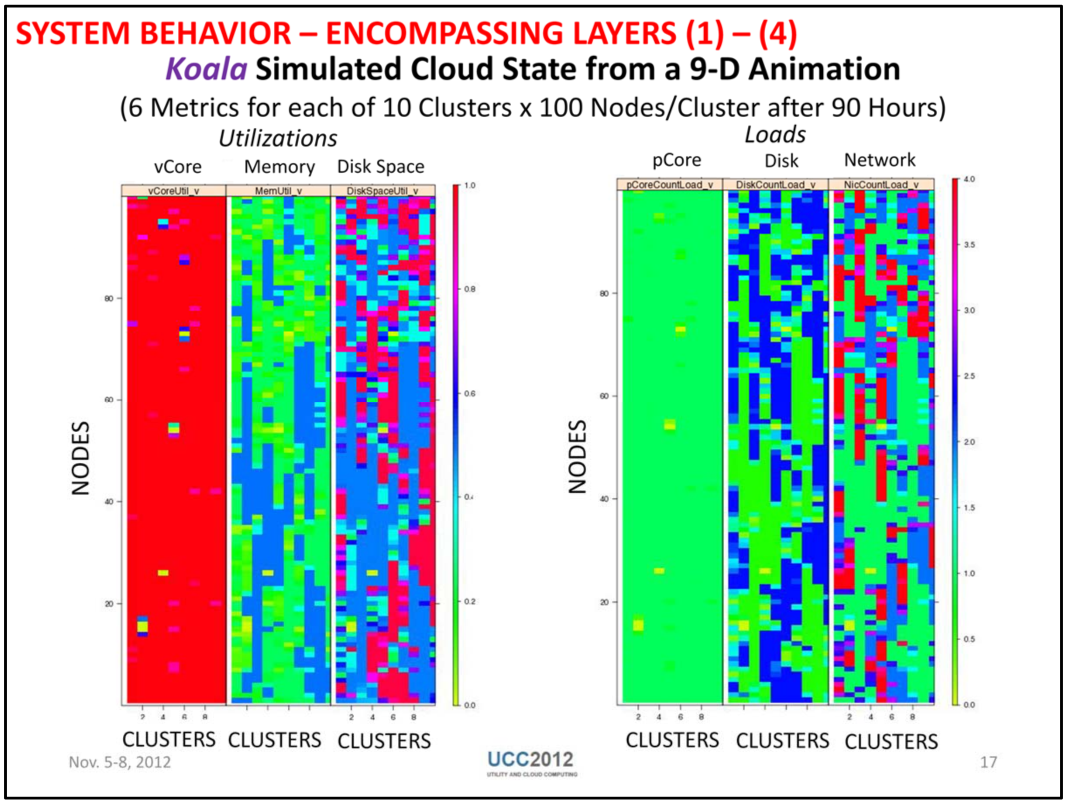


USER START TIME ARRIVES
SELECT USER TYPE, SELECT GEOLOCATION, SELECT # OF REST PERIODS, SELECT THINK PERIOD
TERMINATE INSTANCES RESPONSE
SELECT USER TYPE, SELECT GEOLOCATION, SELECT # OF REST PERIODS, SELECT THINK PERIOD
REFLECTING
REST PERIOD EXPIRES
RESTING
DECREMENT # OF REST PERIODS SELECT # OF RETRIES, SELECT THINK PERIOD
THINK PERIOD ENDS
TERMINATING
SELECT # OF RETRIES, SELECT MIN & MAX VMs, SEND RUN INSTANCE REQUEST
THINK PERIOD EXPIRES
RETRYING
SEND RUN INSTANCE REQUEST
HOLDING PERIOD ENDS
SEND TERMINATE INSTANCES REQUEST
FULLY OR PARTIALLY LAUNCHED
[RETRIES != 0] NERA
HOLDING
SELECT HOLDING PERIOD
REQUESTING
DECREMENT # OF RETRIES, SELECT THINK PERIOD
Randomly describe, reboot and terminate VM instances
[RETRIES == 0 && REST PERIODS != 0] NERA
SELECT REST PERIOD
[RETRIES == 0 && REST PERIODS == 0] NERA
SELECT USER TYPE, GEOLOCATION SELECT # OF REST PERIODS, SELECT THINK PERIOD

Nov. 5-8, 2012 UCC2012 16

Each user executes a lifecycle, as show in this finite-state machine, which has two main paths: (1) the user succeeds in obtaining virtual machines (green) and (2) the user cannot obtain the requested virtual machines and must retry (red). Let me start with the green path. The user arrives, probabilistically selects a type, a geo-location, a persistence (i.e., # of rest periods) and think time, then enters the reflecting state. When the think time expires, the user probabilistically selects a # of retries, a minimum and maximum number of virtual machines of each type needed by the user's type, sends a run-instance request to the cloud and enters the requesting state. Assuming the cloud allocates either the maximum number of virtual machines requested (FULLY LAUNCHED) or at least the minimum number (PARTIALLY LAUNCHED), the user probabilistically selects a holding period and enters the holding state. While holding, the user may probabilistically describe, reboot and terminate virtual-machine instances, which may also crash or be removed by the cloud. When the number of virtual-machine instances falls below the minimum needed, the user may also attempt to obtain additional virtual machines. A user may also decide to increase or decrease the required number of virtual-machine instances during the holding period. When the holding period expires the user sends a terminate-instances request to the cloud controller and enters the terminating state. When the user's instances have been terminated, the user morphs into a new user with a probabilistically selected type, geo-location and so on.

If a user fails to obtain the minimum number of instances, then a number of retries can occur, simulating retries during a business day. If those retries don't succeed, the user can enter a resting period, simulating going home for the day, only to return the next day and try again. If the user's patience is completely exhausted, then the user gives up and morphs into a newly arriving user. Users who give up are considered to be un-served. So the proportion of un-served users is the number of users who gave up over the number of users who arrived during the course of a simulation.



SYSTEM BEHAVIOR – ENCOMPASSING LAYERS (1) – (4)

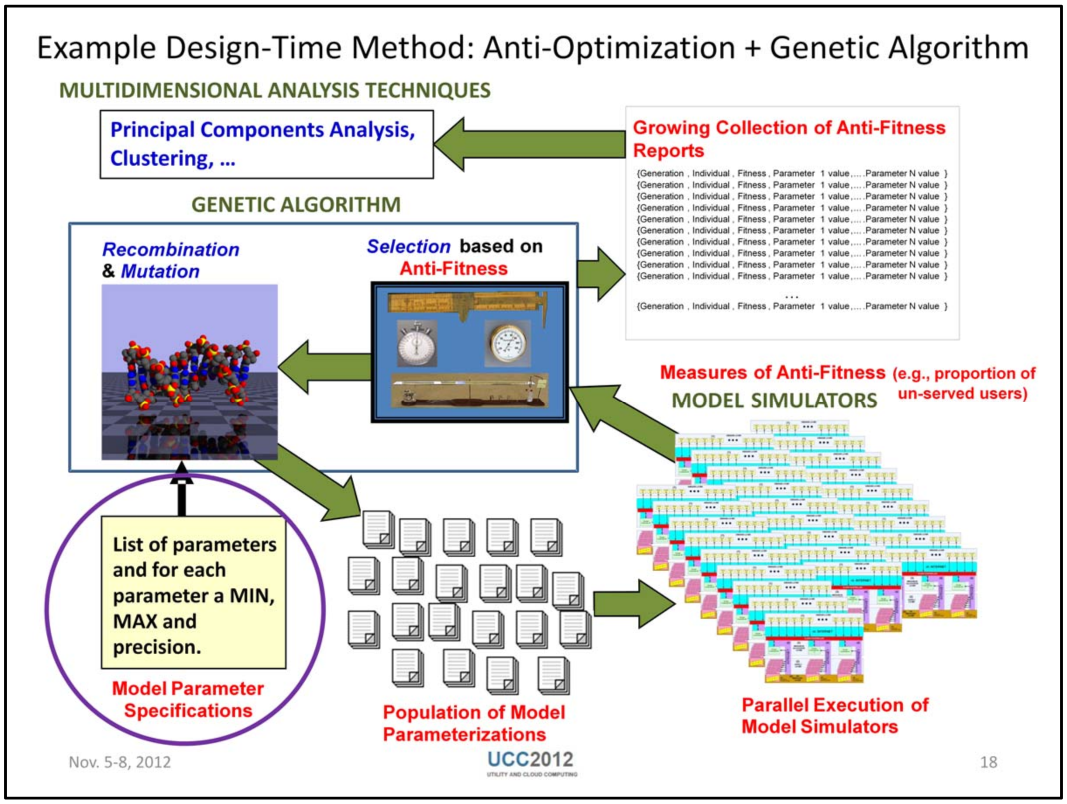
Koala Simulated Cloud State from a 9-D Animation

(6 Metrics for each of 10 Clusters x 100 Nodes/Cluster after 90 Hours)

Nov. 5-8, 2012 UCC2012 UTILITY AND CLOUD COMPUTING 17

To give a sense of Koala's dynamic behavior this slide depicts that 90th hour from a 9-dimensional Koala animation. The panel on the left represents the utilization of three different resources on each node in a cloud comprising 10 clusters each with 100 nodes. The color scale for the panel is just to the right. The first column shows virtual core utilization with the 10 clusters on the x axis and the 100 nodes on the y axis, so each cell represents one of 1000 nodes. The second column in the first panel shows memory usage and the third column shows disk space usage. Clearly, this system is constrained by the availability of virtual cores, which are all nearly 100% utilized. The second panel (which uses a different color scale) represents the load on three different resources on the same 1000 cloud nodes at the same time. In this simulation, physical cores were configured not to allow multiple virtual cores, so the maximum physical core load is one virtual core. The most overloaded resources are the network interfaces, where as many as four virtual network interfaces are mapped to a single physical network interface. This behavior occurs because the cluster algorithms do not consider network interfaces when making a decision about virtual machine allocations. The 9 dimensions of the animation consist of 3 utilization metrics + 3 load metrics + cluster + node within cluster + time.

We have animated Koala simulations with up to 100,000 nodes (100 clusters of 1000 nodes each). Some of these larger animations show an interesting unexpected behavior that we have yet to investigate fully. Likely this will turn out to be an emergent property representing an inherent trait of the system that could not be predicted from analyzing the behavior of the individual components. But that remains to be determined.

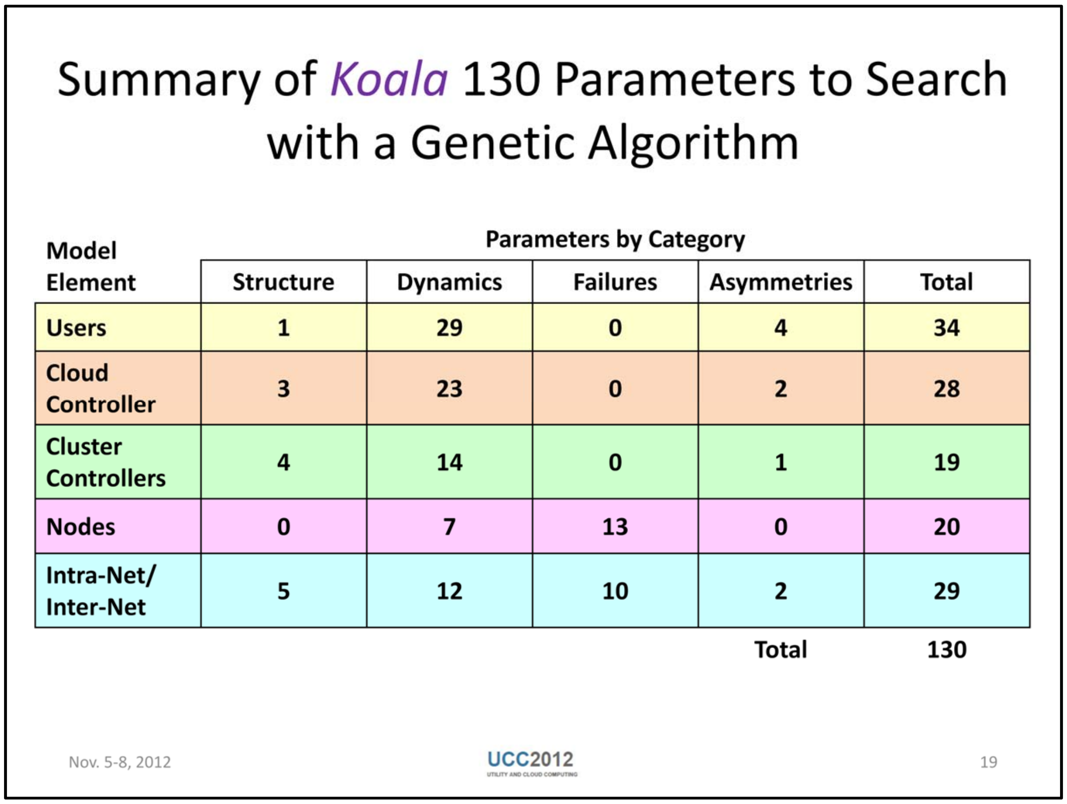


Example Design-Time Method: Anti-Optimization + Genetic Algorithm

MULTIDIMENSIONAL ANALYSIS TECHNIQUES
Principal Components Analysis, Clustering, …
Growing Collection of Anti-Fitness Reports
{Generation , Individual , Fitness , Parameter 1 value,....Parameter N value }
…
GENETIC ALGORITHM
Recombination & Mutation
Selection based on Anti-Fitness
Measures of Anti-Fitness (e.g., proportion of un-served users)
MODEL SIMULATORS
List of parameters and for each parameter a MIN, MAX and precision.
Model Parameter Specifications
Population of Model Parameterizations
Parallel Execution of Model Simulators

Nov. 5-8, 2012 UCC2012 UTILITY AND CLOUD COMPUTING 18

Next, I will outline the Koala parameter space that we will search over: first, giving a general idea of the number and nature of the parameters, and then showing how the GA encodes this information into a chromosome map.



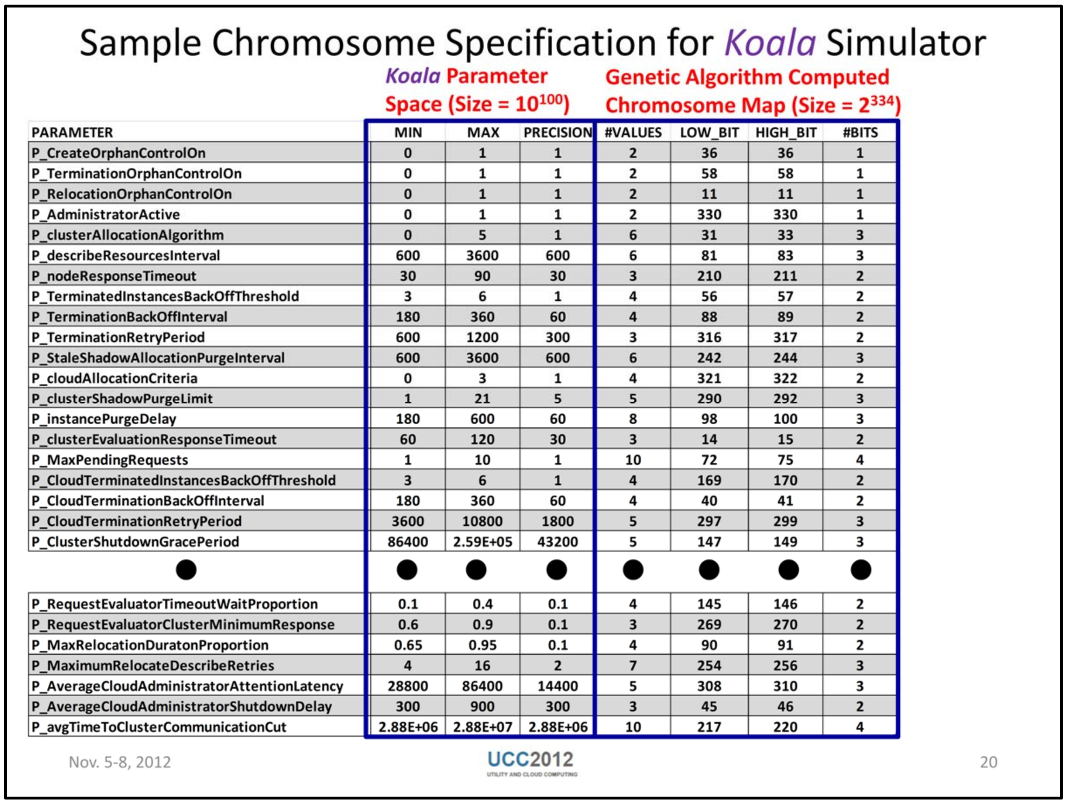
Summary of *Koala* 130 Parameters to Search with a Genetic Algorithm

Model Element	Structure	Dynamics	Failures	Asymmetries	Total
	Parameters by Category				
Users	1	29	0	4	34
Cloud Controller	3	23	0	2	28
Cluster Controllers	4	14	0	1	19
Nodes	0	7	13	0	20
Intra-Net/ Inter-Net	5	12	10	2	29
				Total	130

Nov. 5-8, 2012 UCC2012 UTILITY AND CLOUD COMPUTING 19

Setting aside simulation control parameters, we can focus on parameters associated with each of the five Koala components: (1) users, (2) cloud controller, (3) cluster controllers, (4) nodes and (5) the networks. We selected 130 parameters to search over. We can categorize the parameters by general function. A handful specify structure, such as the distribution of user types, the number of clusters in a cloud, the number of nodes in a cluster, the physical locations of components and the general topology of the simulated Internet. Most parameters relate to dynamics, such as thinking times, retry intervals, back-off periods, number of retries and so on. Most dynamics parameters are associated with the user and the cloud. Some dynamics parameters also appear for other components; for example, to establish network delays, node startup delays and virtual-machine boot times. A few dozen parameters define failures, such as mean-time to failure and failure durations for nodes, node components and communication interfaces, as well as packet-loss probabilities, virtual-machine crash probabilities and so on. The final category of parameters introduce various asymmetries into the model. For example, the distribution of user types may periodically cycle between a small number and a larger number, creating periods of demand that are quite different from each other. Similarly, a cloud may be constructed from a few very large clusters coupled with many very small clusters, rather than the usual assumption of uniform cluster sizes.

Given this selection of 130 parameters, assuming a parameter size of 32 bits, the search space encompasses 10^{1251} states, which is infeasible to search. To reduce the search space, we define the bounds and granularity of each parameter, as I show on the next slide.

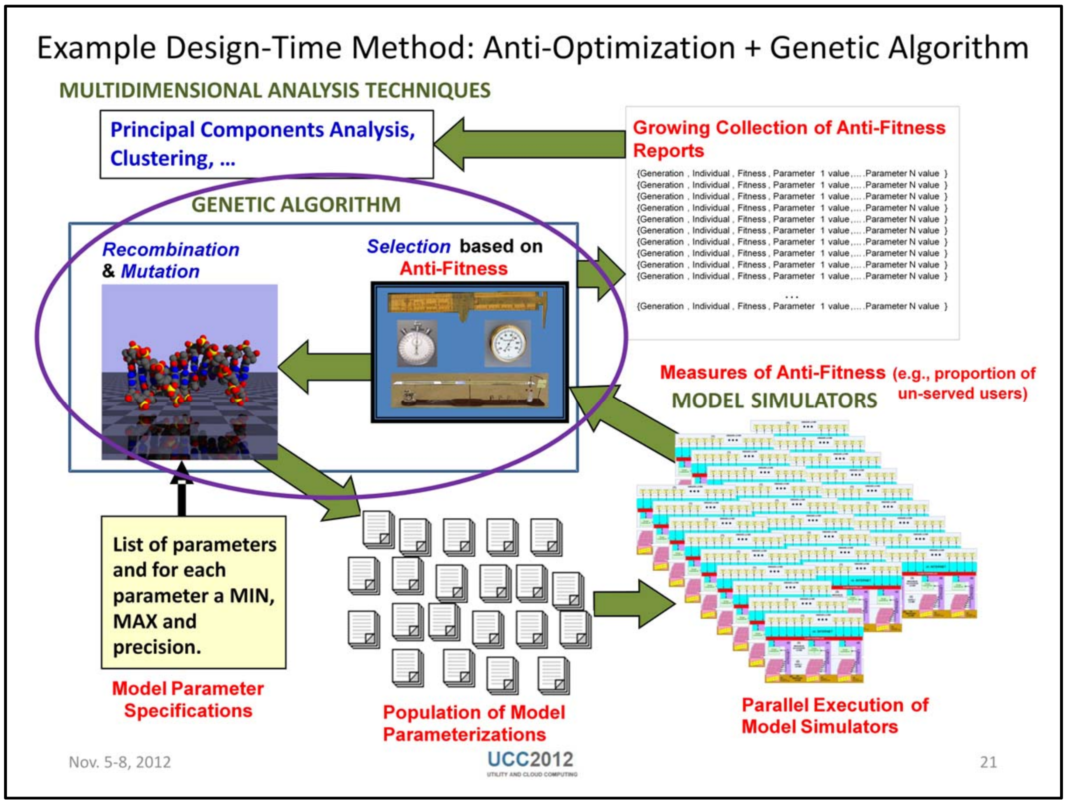


Sample Chromosome Specification for *Koala* Simulator

	Koala Parameter Space (Size = 10^{100})			Genetic Algorithm Computed Chromosome Map (Size = 2^{334})			
PARAMETER	MIN	MAX	PRECISION	#VALUES	LOW_BIT	HIGH_BIT	#BITS
P_CreateOrphanControlOn	0	1	1	2	36	36	1
P_TerminationOrphanControlOn	0	1	1	2	58	58	1
P_RelocationOrphanControlOn	0	1	1	2	11	11	1
P_AdministratorActive	0	1	1	2	330	330	1
P_clusterAllocationAlgorithm	0	5	1	6	31	33	3
P_describeResourcesInterval	600	3600	600	6	81	83	3
P_nodeResponseTimeout	30	90	30	3	210	211	2
P_TerminatedInstancesBackOffThreshold	3	6	1	4	56	57	2
P_TerminationBackOffInterval	180	360	60	4	88	89	2
P_TerminationRetryPeriod	600	1200	300	3	316	317	2
P_StaleShadowAllocationPurgeInterval	600	3600	600	6	242	244	3
P_cloudAllocationCriteria	0	3	1	4	321	322	2
P_clusterShadowPurgeLimit	1	21	5	5	290	292	3
P_instancePurgeDelay	180	600	60	8	98	100	3
P_clusterEvaluationResponseTimeout	60	120	30	3	14	15	2
P_MaxPendingRequests	1	10	1	10	72	75	4
P_CloudTerminatedInstancesBackOffThreshold	3	6	1	4	169	170	2
P_CloudTerminationBackOffInterval	180	360	60	4	40	41	2
P_CloudTerminationRetryPeriod	3600	10800	1800	5	297	299	3
P_ClusterShutdownGracePeriod	86400	2.59E+05	43200	5	147	149	3
●	●	●	●	●	●	●	●
P_RequestEvaluatorTimeoutWaitProportion	0.1	0.4	0.1	4	145	146	2
P_RequestEvaluatorClusterMinimumResponse	0.6	0.9	0.1	3	269	270	2
P_MaxRelocationDuratonProportion	0.65	0.95	0.1	4	90	91	2
P_MaximumRelocateDescribeRetries	4	16	2	7	254	256	3
P_AverageCloudAdministratorAttentionLatency	28800	86400	14400	5	308	310	3
P_AverageCloudAdministratorShutdownDelay	300	900	300	3	45	46	2
P_avgTimeToClusterCommunicationCut	2.88E+06	2.88E+07	2.88E+06	10	217	220	4

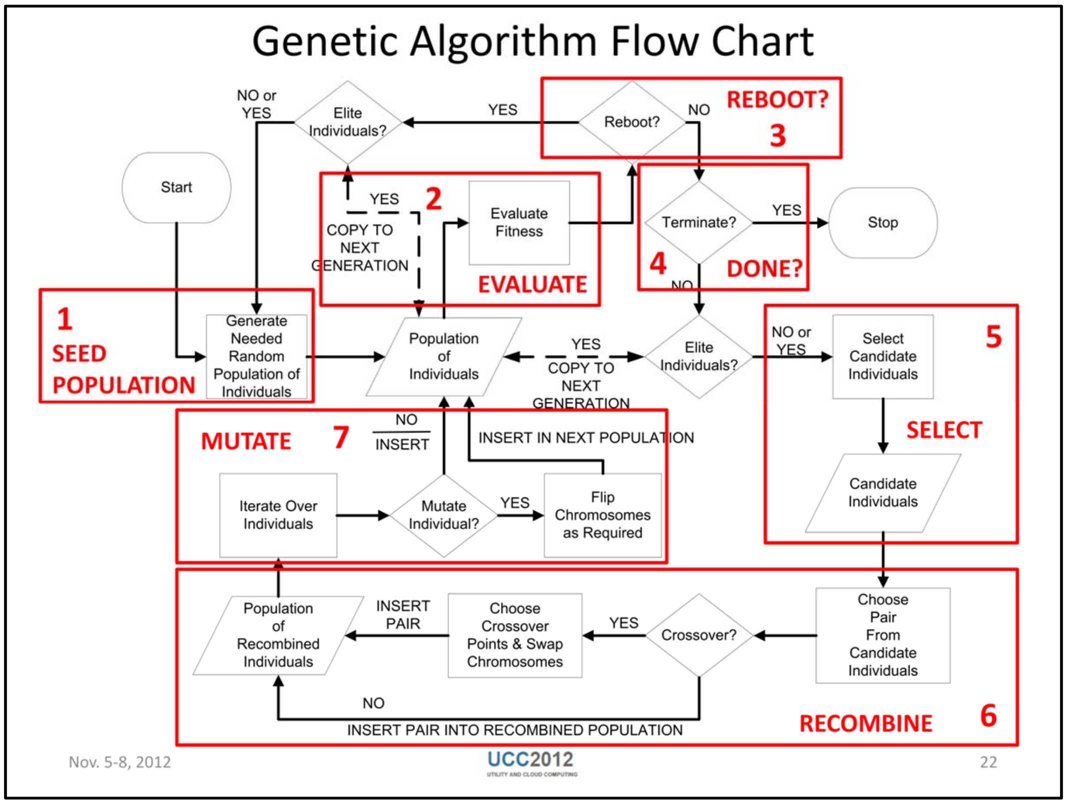
Nov. 5-8, 2012 UCC2012 UTILITY AND CLOUD COMPUTING 20

Here I show an elided list of the 130 Koala parameters over which we will search. Notice that for each parameter, I defined a minimum and maximum value and a precision. These decisions lead to an average of about 6 values per parameter, which means the search space is reduced to (a still formidable) 10^{100}. Once all parameters and associated range and granularities are listed, the GA computes the number of values required for each parameter, which translates into a specific number of bits. The GA then orders the parameters randomly and creates a chromosome map assigning each parameter to specific bit positions. In this example, the chromosome consists of 334 bits, and so the GA's view of the search space size is 2^{334}.



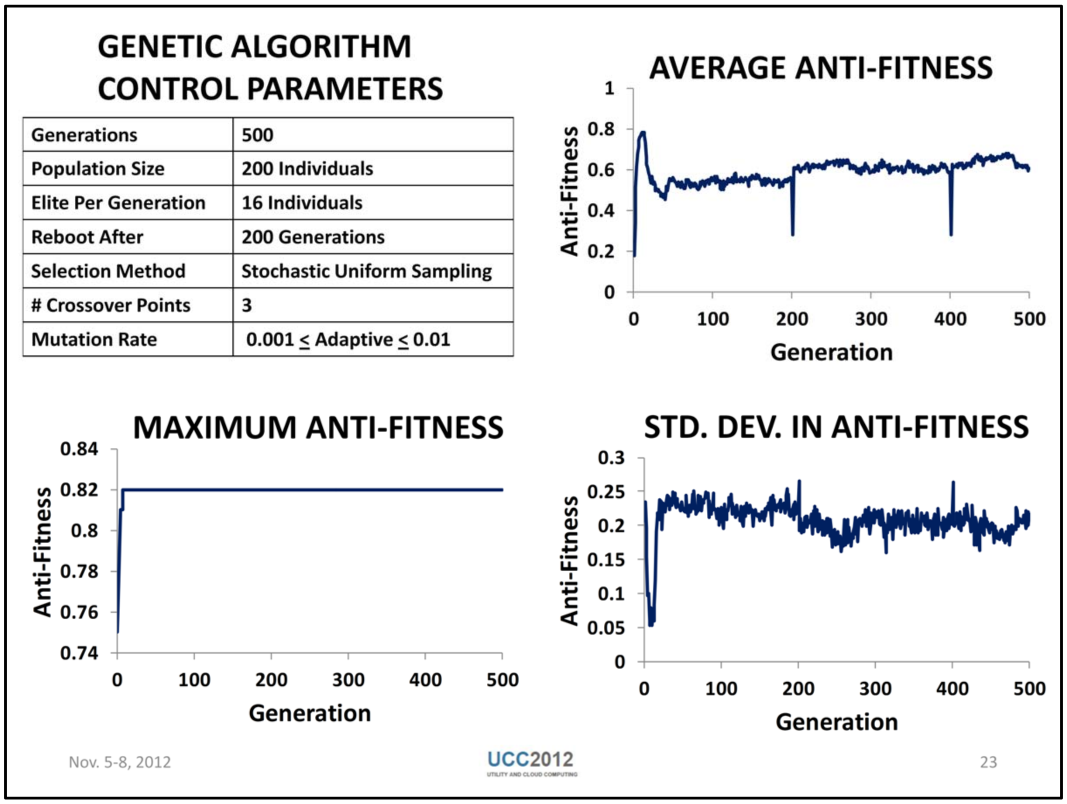
Example Design-Time Method: Anti-Optimization + Genetic Algorithm
MULTIDIMENSIONAL ANALYSIS TECHNIQUES
Principal Components Analysis, Clustering, …
Growing Collection of Anti-Fitness Reports
{Generation , Individual , Fitness , Parameter 1 value,….Parameter N value }
…
GENETIC ALGORITHM
Recombination & Mutation
Selection based on Anti-Fitness
Measures of Anti-Fitness (e.g., proportion of un-served users)
MODEL SIMULATORS
List of parameters and for each parameter a MIN, MAX and precision.
Model Parameter Specifications
Population of Model Parameterizations
Parallel Execution of Model Simulators
Nov. 5-8, 2012
UCC2012
UTILITY AND CLOUD COMPUTING
21

Now, I will describe the general behavior of the GA we used, and illustrate an experiment where the GA explores the Koala parameter space.



The GA can be viewed as a process with 7 steps: one initialization step and 6 cyclic steps, conducted once per generation. Step 1: the GA generations a random chromosome string for each individual in the population. This ensures the starting point of the search is unbiased. Step 2: the GA evaluates the (anti-)fitness of each individual in the population. Here this is accomplished by executing a population of Koala simulations in parallel. When all simulations have finished, fitness evaluation is complete. Step 3: the GA decides whether or not the population needs to be rebooted (i.e., re-randomized), which can occur after a specified number of generations, as determined by the settings of a GA control parameter. If the GA is using elitism, then a specified number of elite individuals (the most fit) are placed unchanged into the next generation, while the remainder are randomly generated. Step 4: the GA decides whether or not to terminate. In our application this occurs after a designated number of generations, but it could be defined to occur when the GA has discovered an answer within a specified tolerance of some target value. Step 5: the GA selects candidate individuals from which to form the next generation. Step 6: the GA randomly selects pairs of individuals from the selected set and recombines them. Our GA uses crossover recombination. A specified number of crossover points is chosen randomly and the bits in the chosen pair of individuals are swapped at those points. Step 7: the GA iterates over each bit in the chromosome of each recombined individual, inverting the bit with some probability, known as the mutation rate.

In general, a GA aims to balance exploitation (implemented via elitism and selection) with exploration (implemented through recombination and mutation). Exploitation tends to emphasize the best solutions found to date, while exploration allows the GA to search in portions of the search space that represent modifications of the best solutions found to date. In the literature, GAs are a form of guided random search. Next, I will show you an example of a GA searching over the parameter space of the Koala simulator.



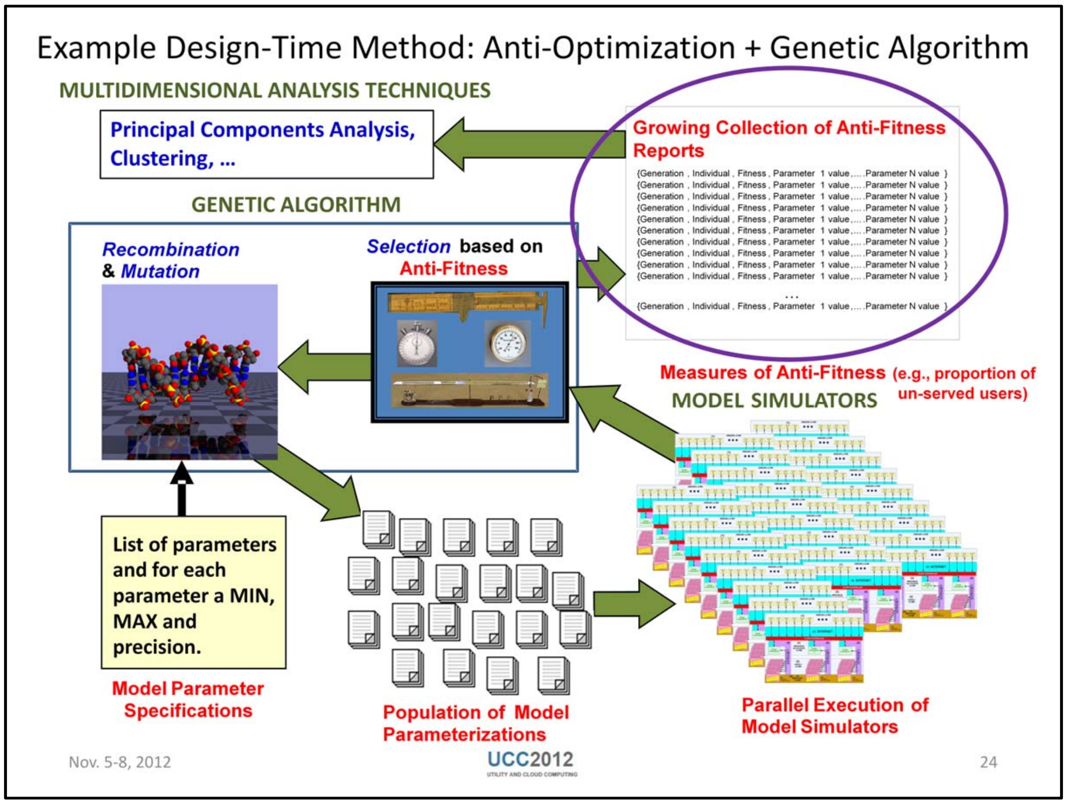
GENETIC ALGORITHM CONTROL PARAMETERS

Generations	500
Population Size	200 Individuals
Elite Per Generation	16 Individuals
Reboot After	200 Generations
Selection Method	Stochastic Uniform Sampling
# Crossover Points	3
Mutation Rate	$0.001 \leq$ Adaptive ≤ 0.01

Nov. 5-8, 2012 UCC2012 UTILITY AND CLOUD COMPUTING 23

This slide shows three graphs depicting the dynamics of the Genetic Algorithm (GA) when steering a population of Koala simulators toward failure scenarios. The table defines the control parameters used for the GA. Given a population of 200 individuals iterated over 500 generations, the GA & Koala explored 10^5 parameter combinations from among the 10^{100} possible. Each graph plots generation (x axis) vs. anti-fitness (y axis), where anti-fitness is the proportion of un-served users. One plot shows the average anti-fitness, which oscillates around 0.55, while showing a gradual upward trend. Since the search starts with a random set of parameters, the average anti-fitness for the first generation is low (just under 0.20). The non-elite population is rebooted twice, at generations 200 and 400, which causes the population's average anti-fitness to drop and then climb back up during the search. The standard deviation in anti-fitness oscillates round 0.20, while exhibiting a gradual downward trend. As the average anti-fitness increases, the standard deviation in anti-fitness decreases, and vice versa. The third plot shows the maximum anti-fitness discovered over all scenarios searched. This starts out at about 0.75 in generation 1 and reaches 0.82 in generation 4. Apparently, the constraints of the problem ensure that at least 18% of the users will be served under any combination of parameters.

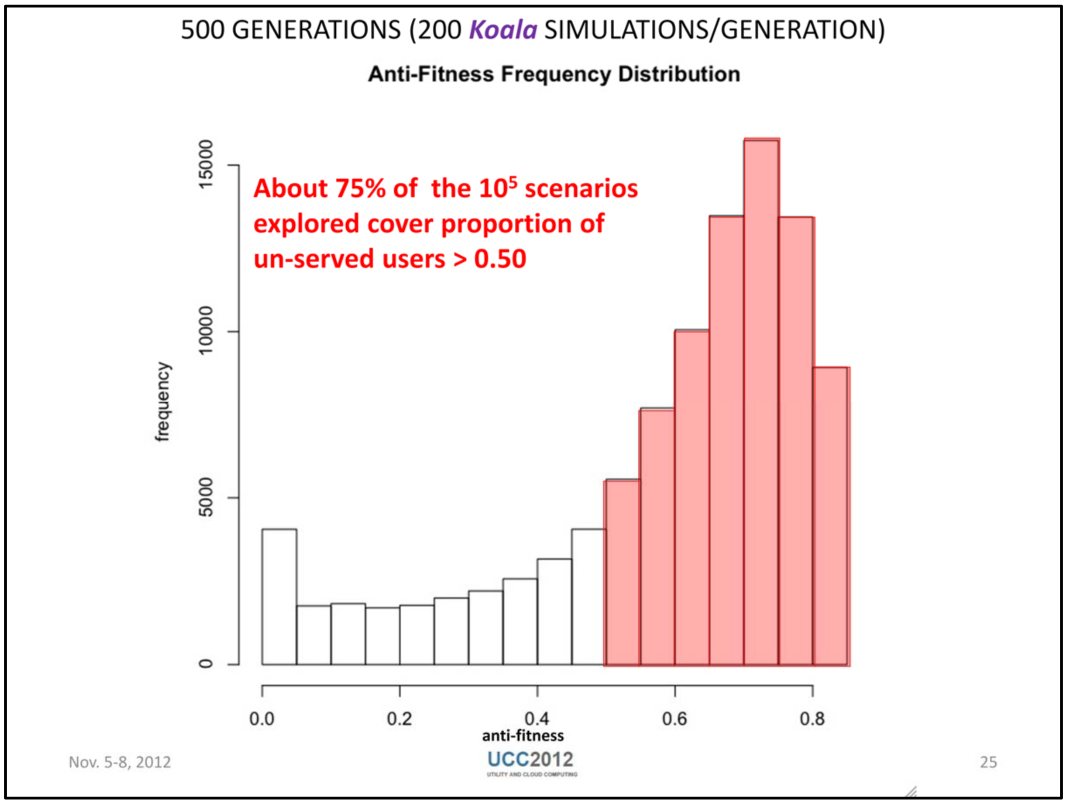
Unlike most searches we have done with this GA the search peaks very early, in generations 6-12, where the population diversity is rather low (standard deviation 0.05). We are investigating whether this outcome is a statistical rarity or whether there is some error.



Example Design-Time Method: Anti-Optimization + Genetic Algorithm

MULTIDIMENSIONAL ANALYSIS TECHNIQUES
Principal Components Analysis, Clustering, …
Growing Collection of Anti-Fitness Reports
{Generation , Individual , Fitness , Parameter 1 value,….Parameter N value }
…
GENETIC ALGORITHM
Recombination & Mutation
Selection based on Anti-Fitness
Measures of Anti-Fitness (e.g., proportion of un-served users)
MODEL SIMULATORS
List of parameters and for each parameter a MIN, MAX and precision.
Model Parameter Specifications
Population of Model Parameterizations
Parallel Execution of Model Simulators
Nov. 5-8, 2012
UCC2012
UTILITY AND CLOUD COMPUTING
24

Finally, I will use the 100,000 anti-fitness values from the tuples collected during the Koala experiment to show that using a GA appears to provide a feasible approach to search for failure scenarios. Since this experiment was completed only recently, we have not undertaken clustering analysis of the data.



500 GENERATIONS (200 *Koala* SIMULATIONS/GENERATION)

This slide plots a frequency distribution of anti-fitness values found in all the scenarios searched. As shown, the preponderance of these scenarios exhibit anti-fitness values above 0.50, which is exactly the range of the scenarios that we hoped to explore. From this, we conclude that using a GA to search the Koala model for failure scenarios appears to be feasible.

We still need to analyze the resulting collection of anti-fitness tuples in order to identify the classes of scenarios that the Koala model exhibits.

National Institute of Standards and Technology

MUCH WORK REMAINS

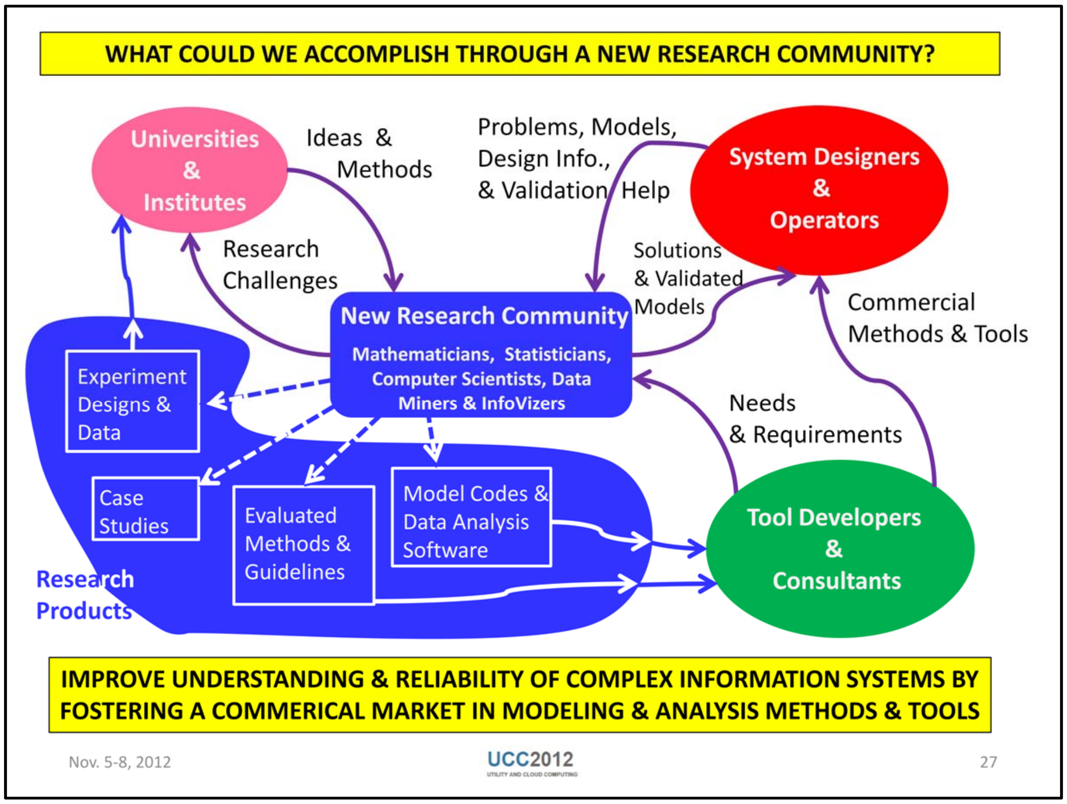
NIST National Institute of Standards and Technology

Image credit– http://2wheeltuesday.com/2011/11/ducati-wants-you/uncle-sam-we-want-you

WE WANT YOU!

Nov. 5-8, 2012 UCC2012 UTILITY AND CLOUD COMPUTING 26

Over the past six years, our small team of NIST researchers has made significant contributions to understanding & predicting behavior in complex information systems, and to methods for identifying performance degradation and failure scenarios. But much work remains. There are many other methods that could be investigated, and on a wide range of systems, both existing and planned. Society needs a wide community of researchers to form around the issues such as those identified in this keynote. That community should include academics & practitioners, as well as government researchers. I am advocating for researchers to join together to develop, investigate and evaluate applied methods that can provide a scientific and engineering understanding of dynamic behavior in the world's complex information systems. The importance of such research will only grow over the near future.



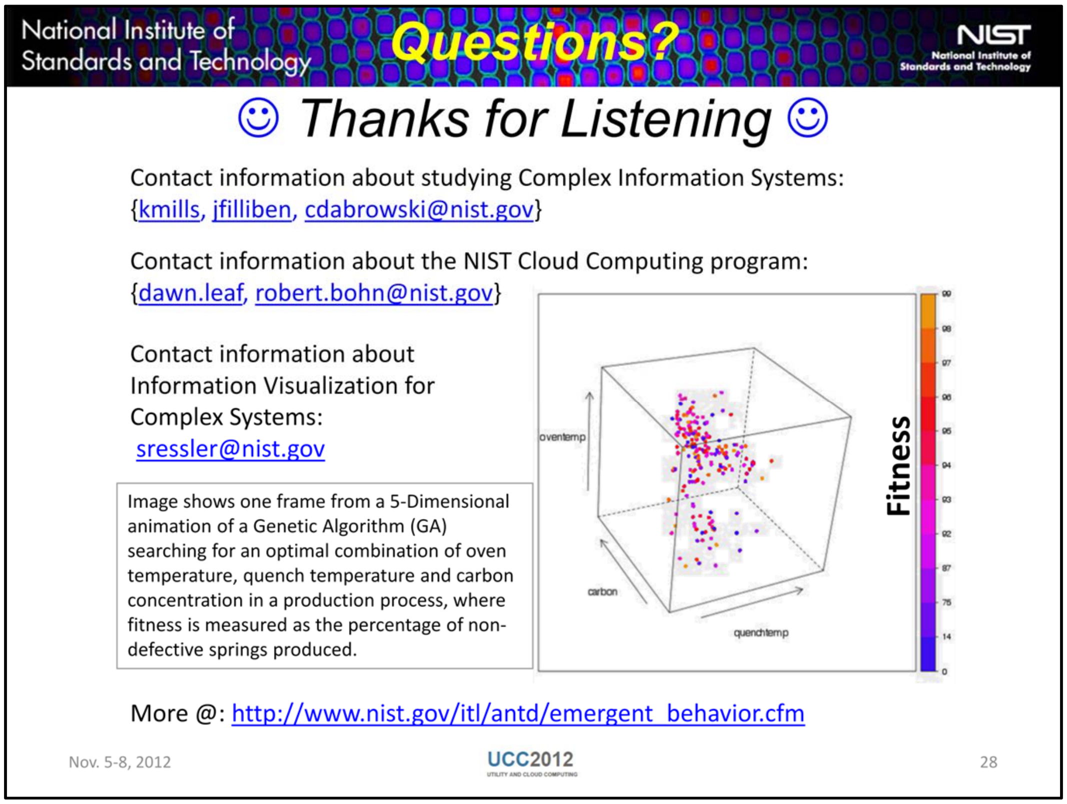
WHAT COULD WE ACCOMPLISH THROUGH A NEW RESEARCH COMMUNITY?

Universities & Institutes
Ideas & Methods
Problems, Models, Design Info., & Validation Help
System Designers & Operators
Research Challenges
Solutions & Validated Models
Commercial Methods & Tools
New Research Community
Mathematicians, Statisticians, Computer Scientists, Data Miners & InfoVizers
Experiment Designs & Data
Needs & Requirements
Case Studies
Evaluated Methods & Guidelines
Model Codes & Data Analysis Software
Tool Developers & Consultants
Research Products

IMPROVE UNDERSTANDING & RELIABILITY OF COMPLEX INFORMATION SYSTEMS BY FOSTERING A COMMERICAL MARKET IN MODELING & ANALYSIS METHODS & TOOLS

Nov. 5-8, 2012 UCC2012 UTILITY AND CLOUD COMPUTING 27

I am advocating formation of a new, multidisciplinary, research community (shown in blue here) that aims to provide evaluated methods & guidelines, model code & data analysis software, experiment designs & data and case studies. The methods & guidelines and model cods & data analysis software would feed a community of commercial tool developers & consultants who would serve a commercial community of system designers & operators. All of this work would aims to improve our understanding of designs and deployments of complex information systems. The research community would be informed by problems, models & design information from system designers and operators and also by needs & requirements form tool developers & consultants. The research community would also be informed by ideas & methods from academe, who would benefit from research challenges identified by the new research community. Over time, as this research community and its stakeholders iterate, the net result would be improved understanding and reliability of complex information systems, and a commercial market for related modeling & analysis methods & tools.



National Institute of Standards and Technology

Questions?

NIST National Institute of Standards and Technology

☺ Thanks for Listening ☺

Contact information about studying Complex Information Systems:
{kmills, jfilliben, cdabrowski@nist.gov}

Contact information about the NIST Cloud Computing program:
{dawn.leaf, robert.bohn@nist.gov}

Contact information about Information Visualization for Complex Systems:
sressler@nist.gov

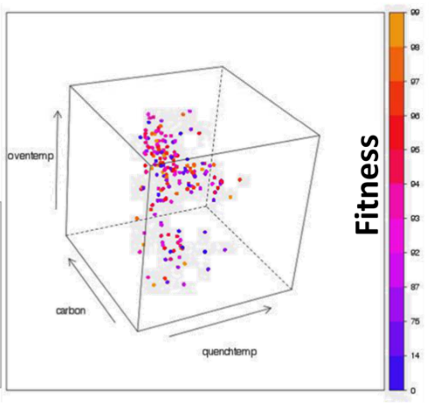


Image shows one frame from a 5-Dimensional animation of a Genetic Algorithm (GA) searching for an optimal combination of oven temperature, quench temperature and carbon concentration in a production process, where fitness is measured as the percentage of non-defective springs produced.

More @: http://www.nist.gov/itl/antd/emergent_behavior.cfm

Nov. 5-8, 2012 UCC2012 UTILITY AND CLOUD COMPUTING 28

Thanks for coming today, and for listening to my address.

Any additional questions?