Dependability for Computer Systems meets Data Analytics

Saurabh Bagchi
School of Electrical and Computer Engineering
Department of Computer Science
Purdue University

Presentation available at: engineering.purdue.edu/dcs1

Roadmap

- **Dependability basics**
  - System design principles
  - Terminology and basic approaches
- **Challenges and current results from**
  - Embedded and mobile networks
  - Computational genomics
- **Dependability in a cellular network** [SRDS-16, Middleware-17, Crowdsense-17, Eurosys-18 (under submission)]
- **Dependability in approximate computing** [PACT-15, CGO-17, ASPLOS-18 (under submission)]
What is Dependable Computing?

• Dependability: Property that the computer system meets its specification despite the presence of faults
  – Faults can be due to natural causes (bugs, defects in hardware), or
  – Maliciously induced (attacks from external or internal sources)

• Terminology
  – **Failure**: Deviation of the delivered service from compliance with the specification
  – **Error**: Part of the system state that has been damaged by the fault and, if uncorrected, can lead to a failure.
  – **Fault**: The adjudged or hypothesized cause of an error

How do We Achieve the Objectives?

- **Applications**
  - Checkpointing and rollback, application replication, software, voting (fault masking), process pairs, robust data structures, recovery blocks, N-version programming,

- **SIFT**
  - Application program interface (API)
  - Middleware

- **Reliable communications**
  - CRC on messages, acknowledgment, watchdogs, heartbeats, consistency protocols

- **Operating system**
  - Memory management, detection of process failures, hooks to support software fault tolerance for application

- **Hardware**
  - Error correcting codes, N_of_M and standby redundancy, voting, watchdog timers, reliable storage (RAID, mirrored disks)
Your Failure is Not My Failure

- Crash
- Omission
- Timing
- Incorrect Computation
- Byzantine (malicious)

3 Most Important Things in Dependability

1. Redundancy
2. Redundancy
3. Redundancy

Means of achieving dependability

- How much redundancy?
- Where to put the redundancy?
- How to validate the redundant operation?
Roadmap

- Dependability basics
  - System design principles
  - Terminology and basic approaches
- Challenges and current results from
  - Embedded and mobile networks
  - Computational genomics
- Dependability in a cellular network [SRDS-16, Middleware-17, Crowdsense-17, Eurosys-18 (under submission)]
- Dependability in approximate computing [PACT-15, CGO-17, ASPLOS-18 (under submission)]

Embedded and Mobile Networks

- Challenges
  - Systems have fundamental resource constraints and are often deployed in unprotected or uncertain environments
  - Constraints include: Energy, Bandwidth, Untrusted nodes, Disconnected networks
  - Nodes have low-end microcontrollers and lightweight OS without security protection
Embedded and Mobile Networks

- Opportunities
  - Fewer modes of user interaction
  - Single purpose
  - Dense deployment

- Some active research directions
  1. Distributed monitoring for mobile networks (such as, vehicular networks)
  2. Cellular radio access problems
  3. Record and replay for problem diagnosis

A Significant Result

- **TARDIS**: A software-only approach for deterministic record and replay of embedded nodes\[1, 2\]
- Basic idea of any record-and-replay mechanism:
  1. Record detailed execution trace as application is executing on node in situ
  2. Use the trace to replay the execution and debug the problem
- Records all sources of non-determinism and compresses different traces in a domain-specific manner
- **Result**: Log growth = 1.5 KB/s – 88% reduction ⇒ Flash on Amazon Echo Dot can hold 1 month of blackbox information
- Need fine-grained monitoring information to debug subtle timing bugs

A Significant Result

- Need fine-grained monitoring information to debug subtle timing bugs

![Graph showing time vs. data received]

- Given rise to a community building debugging tools for low-end embedded devices and networks [Minerva-Sensys13, Flocklab-IPSN13, D2-RTSS13, Thiele-et al.-EWSN17, …]

Data Analytics and Dependability
(In Embedded and Mobile Networks)

<table>
<thead>
<tr>
<th>Current state of practice</th>
<th>Very limited</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level of sophistication of data analysis needed</td>
<td>Simple</td>
</tr>
</tbody>
</table>

- Examples of promising convergence of the two
  1. Learn the pattern of traffic between two interacting devices – use that for anomaly detection
  2. Learn the pattern of sensed values in a region or in a time period – use that for optimal compression
Roadmap

• Dependability basics
  – System design principles
  – Terminology and basic approaches
• Challenges and current results from
  – Embedded and mobile networks
  – Computational genomics
• Dependability in a cellular network [SRDS-16, Middleware-17, Crowdsense-17, Eurosys-18 (under submission)]
• Dependability in approximate computing [PACT-15, CGO-17, ASPLOS-18 (under submission)]

Computational Genomics

• Phenomenal growth in amount of genomic (and epigenomic) data which has to be processed for insights
• Sequencing instruments are error prone – redundancy in reads and algorithmic tricks have to correct for errors
• Common dependability challenges:
  – Scalability, scalability, scalability
  – Fragile code bases
  – Lack of efficient cyberinfrastructures
Computational Genomics

- Opportunities
  - Some emerging standardized building blocks
  - Some embarrassingly parallel parts to many algorithms
  - Good metrics for judging accuracy of results and benchmarking

- Some active research directions
  - *Error correction*: Methods to correct various kinds of errors in sequencing reads
  - *Applying standard distribution techniques*: From graph algorithms or string matching for example
  - *Standardized workflows and testing*: Reduces incompatibilities among multiple software packages and allows us to judge quality of results

A Significant Result

- Across several subdomains, tools containing a recurring set of building blocks or kernels which become the constructs.


Performance Comparison of SARVAVID over Vanilla Applications

<table>
<thead>
<tr>
<th>Application</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLAST</td>
<td>2.5</td>
</tr>
<tr>
<td>MUMmer Application</td>
<td>3.0</td>
</tr>
<tr>
<td>E-MEM</td>
<td>2.0</td>
</tr>
<tr>
<td>SPAdes</td>
<td>1.5</td>
</tr>
<tr>
<td>SGA</td>
<td>1.0</td>
</tr>
</tbody>
</table>

- Application in Sarvavid
- Vanilla Application

Data Analytics and Dependability (In Computational Genomics)

<table>
<thead>
<tr>
<th></th>
<th>Current state of practice</th>
<th>Level of sophistication of data analytics needed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Primitive</td>
<td>High but needs to be scalable as well</td>
</tr>
</tbody>
</table>

- Examples of promising convergence of the two
  1. Learn the pattern of errors in a particular instrument and particular configuration to decide how best to correct errors
  2. Learn the loading pattern of different tasks so as to optimally distribute load
  3. Complex multi-dimensional pattern in epigenomic markers to tell why some genes are repressed
Roadmap

- Dependability basics
  - System design principles
  - Terminology and basic approaches
- Challenges and current results from
  - Embedded and mobile networks
  - Computational genomics
- Dependability in a cellular network [SRDS-16, Middleware-17, Crowdsense-17, Eurosys-18 (under submission)]
- Dependability in approximate computing [PACT-15, CGO-17, ASPLOS-18 (under submission)]

Motivation

- Mobile devices consider cellular network as “dumb pipe”
  - Connectivity is not always reliable
  - Can only react after connectivity is degraded
- With cooperation between device and network, failures can be managed better
**TANGO**

- Framework that enables real-time cooperation between mobile device and cellular network
  - Many services can be built on top
- Service performs real-time data analysis to alert device of certain events
  - Alert streaming application before user enters congested area
- Device/application decides how to mitigate problem
  - Streaming application: pre-cache more content
  - Switch to a different carrier
Pre-caching Service

- Sends alert to application before connectivity degradation
  - Predicting user location
  - Monitor network load on predicted locations
- Applications
  - Audio/Video streaming
  - GPS navigation
  - Web browsing
Streaming Application – Current Practice

- Mobile streaming applications limit download rate
  - To conserve user’s bandwidth and energy
  - Usually by limiting buffer size
- Connectivity degradation (e.g., due to congestion) results in playback disruption
- Ideal buffering strategy:
  - Small buffer when connectivity is good
  - Large buffer when connectivity is bad
- With current systems, can only react after connectivity degraded
- With TANGO, application increases buffer size and/or reduce bit-rate

Pre-caching Service – Overview

- Offline phase
  - Location predictor training
- Online phase
  - User location prediction
  - Network load monitoring

Legend
- Online Phase
- Offline Phase
AppStreamer: Reducing mobile applications’ storage requirements through predictive streaming

Motivation

• Demand for storage has been growing faster than storage capacity of smartphones
  – 2016 survey found amount of content stored grew 55% over 10 months
  – Many popular games are 1-4 GB
  – Smartphone capacity roughly doubles every 2.5 years
  – Users need to uninstall some apps to make space for new apps
• Current practice: all or none
  – Installation also takes long time
Available Solutions to Storage Crunch

- Storing apps on the cloud
  - Unacceptable delay even with good connectivity
  - Application usage is difficult to predict
- Running apps on the cloud and streaming only video (thin client)
  - ~100ms input latency unacceptable in interactive applications such as games
  - High bandwidth consumption

Solution Approach

- Intuition: large apps only access small portion of their files
- AppStreamer: predictive application streaming
  - Collect data about file accesses from multiple users
  - Build a model that predicts future file accesses based on recent behavior
AppStreamer Overview

Offline Phase (training)
- Trace data
- Partitions
- Equivalent Partitions
- Superblocks
- Continuous-time Markov Chain Model

Online Phase
- Block Fetcher
- Predicted blocks
- Continuous-time Markov Chain Model
- Real-time file accesses
- Application

Key Requirements
• Work for all applications without source code
  – Transparent to the applications
  – Blocks need to be on local storage before they are read
• Does not lower user experience by introducing delays
  – Accurate model (high recall)

AppStreamer Components
1. File access capture (offline & online phases)
2. File access prediction model (online phase)
3. Data block fetcher (online phase)
File Access Prediction Model (1)

• Requirements
  – Temporally based
  – Low overhead
  – Probabilistic

• Continuous-Time Markov Chain (CTMC)
  – Captures transition between states & duration spent
  – Need to define “state”

File Access Prediction Model (2)

• Idea 1: block = state
  – 1 GB game has 250,000 blocks. Need to store $250,000^2 = 62.5$ billion transition probabilities.

• Idea 2: read call = state
  – Similar reads considered distinct
  – One large read can be separated into multiple reads in many ways
**Block Grouping (1)**

- Input: File access traces from multiple runs
- Three steps
  1. Partition
  2. Equivalent Partition
  3. Superblock

**Markov Chain**

- State = superblock
- Prediction done using depth-first search
  - Bounded by probability ($p_{stop}$) and lookahead time ($L$)
  - User playing speed taken into account by adjusting $L$
- Blocks with probability $\geq p_{download}$ are put in download queue
- Single Markov model can capture different control paths taken by different users
Data Block Fetcher

- Speculative blocks
  - Fetches predicted blocks in background
- Application-requested blocks (cache miss)
  - Read system call will block
  - Fetches immediately from cloud storage server

Evaluation: Dead Effect 2

- First-person shooter
- Single player
- Gameplay divided into linear levels (maps)
- 1.09 GB in size
User Study Setup

- Cloud storage server with 17.4 Mbps network speed
- Nexus 6P running Android 6.0.1
- User plays game for 20-30 minutes
  - With and without AppStreamer
- Fill questionnaire to report user experience

Dead Effect 2 User Study Results (1)

- 23 participants

![Pie charts showing skill level, loading time, overall user experience, and delay during a level.](image-url)
Dead Effect 2 User Study Results (2)

- 336.2 KB cache miss per run on average
  - Translates to 0.15 second of delay (out of ~30 minutes)
- 87.16% lower storage requirements

Comparison with Cloud Gaming

- GamingAnywhere [2]
  - Android emulator Nox on cloud server
  - GamingAnywhere client on smartphone

Concluding Insights

• Cooperation between mobile devices and cellular network can enable highly reliable operation
  – Reliable streaming of content
  – Reducing streaming of mobile applications
• Such cooperation can reduce pressure on constrained resources
  – Storage on the device
  – Wireless bandwidth

Roadmap

• Dependability basics
  – System design principles
  – Terminology and basic approaches
• Challenges and current results from
  – Embedded and mobile networks
  – Computational genomics
• Dependability in a cellular network [SRDS-16, Middleware-17, Crowdsense-17, Eurosys-18 (under submission)]
• Dependability in approximate computing [PACT-15, CGO-17, ASPLOS-18 (under submission)]
Performance Errors versus Accuracy

Can tolerate some imprecision

Image Processing
Media Applications
Machine Learning

Data Analytics
Computer Vision
Scientific Simulations

Simple Examples of Approximation Techniques

for (i = 0; i < n; i = i + approx_level)
result = compute_result();

Loop Perforation

for (i = 0; i < n; i++)
if (0 == i % approx_level)
    cached_result = result = compute_result();
else
    result = cached_result;

Memoization
Output quality degradation in Sobel

0% Quality Loss  5% Quality Loss  10% Quality Loss

10% Quality loss is nearly indiscernible to the eye and yet provides 57% energy savings

Approximate computing: Trade accuracy for energy saving or computation speedup

Adjust knobs to control the approximation-levels of computation

Speedup and Energy Reduction

Accuracy

LULESH – A Hydrodynamic Simulation
Assumption of Monolithic Execution

- The general approach has been to have single approximation configuration throughout the entire execution

Phases in super-loop computations

```c
1  elements = A set of elements to simulate
2  While(state->stable == false)
3  {
4    increment_simulation_time();
5    forces_on_elements();
6    acceleration_of_elements();
7    velocity_of_elements();
8    position_of_elements();
9    strain_of_elements();
10   calculate_timeconstraints();
11   state = get_current_state();
3  }
```
Workflow of OPPROX

Phase-specific QoS Characteristics

LULESH

Bodytrack
Phase-specific Speedup Characteristics

![Graphs showing speedup characteristics for LULESH and Bodytrack.](image)

Input-Dependent Behavior

![Graphs showing QoS degradation and speedup for Bodytrack and LULESH.](image)
Evaluation of the Models

Speedup Prediction

QoS degradation Prediction

Speedup obtained by Opprox

Phase-specific approximation is more attractive when operating under small error budget
Approximation in Video Processing

- **Fundamental challenge #1:** Best settings of approximate computations that can provide acceptable results *for the given input*
  - Optimal approximation setting is dependent on the content of the video
  - Empirically: Even different frames within the same video should have different values of approximation settings
- PSNR plot
- Actual video 1
- Approximate video 1 (acceptable)
- Actual video 2
- Approximate video 2 (unacceptable)

Approximation in Video Processing

- **Fundamental challenge #2:** Search space of approximation settings is large
  - Say: *k* stages in pipeline (small, say 2-5), *n* approximation settings for each stage (large, *n* ≥ 10), then search space = \(\Theta(n^k)\)
  - Some model needs to be developed for efficiently searching through the space
Our Solution: VideoChef

- Uses optimization techniques to find the best suitable settings for various approximation techniques inside video processing kernels
- Uses small-sized canary input to guide choice of approximation settings during optimization search
- Builds a prediction model for mapping the error from the canary input to that with the original input
- Online, VideoChef performs efficient search through approximation search space and bounds the search so that benefit of approximating the computation is realized
- VideoChef is the first technique that can apply tunable approximation algorithms to a streaming application in a manner that the benefit of approximation (minimization of computation cost) is actually realized
Modeling Error

- **Error mapping model**: Characterize the relation between error in the canary output and error in the full output, for the same approximation levels
  - Model C: Knows error in C, predicts error in F
  - Model CA: Knows error in C and approximation level in effect, predicts error in F
  - Model CAV: Knows error in C, approximation level in effect, and video characteristics, predicts error in F

Triggering Search

- MPEG-4 and many other video formats define three main types of frames: I, P, and B frames.
  - An I-frame uses intra-prediction meaning, the P- and the B-frames use inter-prediction
  - When to insert an I-frame: a big difference in the frame triggers the insertion of a new I-frame, since inter-coding will give almost as long a code as intra-coding.
  - In VideoChef, we leverage this observation and do a single search for a group of pictures, where a group is demarcated by I-frames at its two ends.
Evaluation

- 106 YouTube MPEG-4 videos
  - Collected from 8 different categories to cover a spectrum of different motion and color artifacts in the frames
- The user-provided acceptable video quality threshold is 30 dB, which is considered a typical lower bound for lossy image and video compression
- The different comparable protocols are:
  1. Exact computation,
  2. static approximation
  3. IRA [PLDI 2016]
  4. VideoChef Greedy
  5. VideoChef GA
  6. Oracle

End-to-End Workflow Results

- VideoChef increases speedup by 28.7% over exact computation and is within 10.3% of the Oracle
- It outperforms both static approximation and IRA, by 22.4% and 33.7%
- Advantage exists for all the content categories; greatest savings in category 5 (movie trailers)
- Search overhead of VideoChef is small but it finds more aggressive approximations
- Counter-intuitively, for IRA, the number of executed instructions is higher than for exact computation, by 4.9%
Data Analytics in Systems: A Cautionary Tale

- We need data analytics (no prediction is really bad)
- But model CA does nearly as well as CAV
- CAV model parameters much harder to obtain at runtime

Concluding Insights

- Human perception is tolerant to errors
  - Allows for approximation and thus savings in energy and runtime
  - Approximation has to be done in a content-specific way
- But approximate computation needs to be agile
  - Needs to keep quality of output above user-specified threshold
  - Needs efficient mechanism to extract features from content
  - Needs efficient mechanism to search through large space of approximation settings
- VideoChef: First small step for approximation in streaming applications, in a content-dependent manner
Concluding Insights: Dependability and Data Analytics

- Dependability involves handling natural and malicious failures
- Handling involves: prevent, detect, diagnose, contain, repair
- Rule-bases, exact solutions giving way to data analytic solutions
  - Too much data
  - Too much noise
  - Too many modes of interaction
  - Incomplete observability into some software components

Concluding Insights: Dependability and Data Analytics

- Data analytic solutions themselves have to be scalable
- Solutions have to make adjustable trade-offs between false positive and false negative errors
- Different domains provide different constraints and opportunities for the dependability solutions
  - Embedded and mobile networks
  - Computational genomics
**Take-Aways: More Philosophical**

- Explore at the point of greatest curiosity
  - But, beware of rabbit holes
  - Learn from what was tried but did not work
- Do not be intimated by volume of prior work
  - Question assumptions underlying the work: Have they changed due to world moving on/technological changes
  - Experiment, do not be an armchair researcher – in the small first, and see if the prior story still holds
  - Learn from your own mistakes
- Socialize your research
  - Discuss with your peers, not just in your immediate group
  - Cooperate - open source code, brainstorm ideas, give feedback on other’s drafts – successful students often have a supportive ecosystem around them

---

**It Takes a Village**

- **Purdue**: [Faculty] Milind Kulkarni, Mathias Payer, He Wang, [Students] Abe Clements, Kanak Mahadik, Matt Creti, Nawanol Theera, Chris Wright
- **Argonne National Lab**: Folker Meyer
- **AT&T Labs**: Kaustubh Joshi, Rajesh Panta
- **Google**: Greg Bronevetsky
- **Georgia Tech**: Mostafa Ammar, Ellen Zegura
- **UIUC**: Sasa Misailovic