CS 6410: ADVANCED SYSTEMS
KEN BIRMAN

Fall 2012

A PhD-oriented course about research in systems

About me...

- My research is focused on "high assurance"
  - In fact as a graduate student I was torn between machine learning in medicine and distributed systems
  - I've ended up working mostly in systems, on topics involving fault-tolerance, consistency, coordination, security and other kinds of high-assurance
- My current hot topics?
  - Cloud-scale high assurance via platform and language support (often using some form of machine learning)
  - Using the cloud to monitor/control the smart power grid
- ... but CS6410 is much broader than just "Ken stuff"

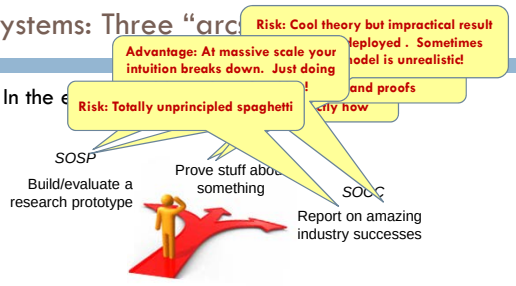
Goals for Today

- What is CS6410 "about"?
  - What will be covered, and what background is assumed?
  - Why take this course?
  - How does this class operate?
  - Class details
- Non-goal: We won't have a real lecture today
  - This is because our lectures are always tied to readings

Coverage

- The course is about the cutting edge in computer systems – the topics that people at conferences like ACM Symposium on Operating Systems Principles (SOSP) and the Usenix Conference on Operating Systems Design and Implementation (OSDI) love
- We look at a mix of topics:
  - Classic insights and classic systems that taught us a great deal or that distilled key findings into useable platform technologies
  - Fundamental (applied theory) side of these questions
  - New topics that have people excited right now

Systems: Three "arc



- In the e
- Today, these lines are more and more separated
- Some people get emotional over which is best!

My work blends theory and building

- This isn't unusual, many projects overlap lines
- But it also moves me out of the mainstream SOSP community: I'm more of a "distributed systems" researcher than a "core systems" researcher
- My main interest: *How should theories of consistency and fault-tolerance inform the design of high-assurance applications and platforms?*

Questions this poses

- Which theory to use? We have more than one theoretical network model (synchronous, asynchronous, stochastic) and they differ in their "power"
- How to translate this to a provably sound systems construct and to embed that into a platform (we use a model shared with Lamport's Paxos system)
- Having done all that, how to make the resulting system scale to run on the cloud, perform absolutely as fast as possible, exhibit stability... how to make it "natural" to use and easy to work with...

Current passion: my new Isis^2 System

- C# library (but callable from any .NET language) offering replication techniques for cloud computing developers
- Based on a model that fuses virtual synchrony and state machine replication models
- Research challenges center on creating protocols that function well despite cloud "events"

Elasticity (sudden scale changes)	Long scheduling delays, resource contention
Potentially heavily loads	Bursts of message loss
High node failure rates	Need for very rapid response times
Concurrent (multithreaded) apps	Community skeptical of "assurance properties"

Isis^2 makes developer's life easier

Benefits of Using Formal model	Importance of Sound Engineering
Formal model permits us to achieve correctness	Isis^2 implementation needs to be fast, lean, easy to use
Isis^2 is too complex to use formal methods as a development too, but does facilitate debugging (model checking)	Developer must see it as easier to use Isis^2 than to build from scratch
Think of Isis^2 as a collection of modules, each with rigorously stated properties	Seek great performance under "cloudy conditions"
	Forced to anticipate many styles of use

Isis^2 makes developer's life easier

10

```
Group g = new Group("myGroup");
g.ViewHandlers += delegate(View v) {
    Console.Title = "myGroup members: "+v.members;
};
g.Handlers[UPDATE] += delegate(string s, double v) {
    Values[s] = v;
};
g.Handlers[LOOKUP] += delegate(string s) {
    Reply(Values[s]);
};
g.Join();

g.Send(UPDATE, "Harry", 20.75);

List<double> resultlist = new List<double>;
nr = g.Query(LOOKUP, ALL, "Harry", EOL, resultlist);
```

- First sets up group
- Join makes this entity a member. State transfer isn't shown
- Then can multicast, query. Runtime callbacks to the "delegates" as events arrive
- Easy to request security (g.SetSecure), persistence
- "Consistency" model dictates the ordering aseen for event upcalls and the assumptions user can make

Isis^2 makes developer's life easier

11

```
Group g = new Group("myGroup");
g.ViewHandlers += delegate(View v) {
    Console.Title = "myGroup members: "+v.members;
};
g.Handlers[UPDATE] += delegate(string s, double v) {
    Values[s] = v;
};
g.Handlers[LOOKUP] += delegate(string s) {
    Reply(Values[s]);
};
g.Join();

g.Send(UPDATE, "Harry", 20.75);

List<double> resultlist = new List<double>;
nr = g.Query(LOOKUP, ALL, "Harry", EOL, resultlist);
```

- **First sets up group**
- Join makes this entity a member. State transfer isn't shown
- Then can multicast, query. Runtime callbacks to the "delegates" as events arrive
- Easy to request security (g.SetSecure), persistence
- "Consistency" model dictates the ordering seen for event upcalls and the assumptions user can make

Isis^2 makes developer's life easier

12

```
Group g = new Group("myGroup");
g.ViewHandlers += delegate(View v) {
    Console.Title = "myGroup members: "+v.members;
};
g.Handlers[UPDATE] += delegate(string s, double v) {
    Values[s] = v;
};
g.Handlers[LOOKUP] += delegate(string s) {
    Reply(Values[s]);
};
g.Join();

g.Send(UPDATE, "Harry", 20.75);

List<double> resultlist = new List<double>;
nr = g.Query(LOOKUP, ALL, "Harry", EOL, resultlist);
```

- First sets up group
- **Join makes this entity a member. State transfer isn't shown**
- Then can multicast, query. Runtime callbacks to the "delegates" as events arrive
- Easy to request security (g.SetSecure), persistence
- "Consistency" model dictates the ordering seen for event upcalls and the assumptions user can make

Isis² makes developer's life easier

13

```
Group g = new Group("myGroup");
g.ViewHandlers += delegate(View v) {
    Console.Title = "myGroup members: "+v.members;
};
g.Handlers[UPDATE] += delegate(string s, double v) {
    Values[s] = v;
};
g.Handlers[LOOKUP] += delegate(string s) {
    Reply(Values[s]);
};
g.Join();

g.Send(UPDATE, "Harry", 20.75);

List<double> resultlist = new List<double>;
nr = g.Query(LOOKUP, ALL, "Harry", EOL, resultlist);
```

- First sets up group
- Join makes this entity a member. State transfer isn't shown
- **Then can multicast, query. Runtime callbacks to the "delegates" as events arrive**
- Easy to request security (g.SetSecure), persistence
- "Consistency" model dictates the ordering seen for event upcalls and the assumptions user can make

Isis² makes developer's life easier

14

```
Group g = new Group("myGroup");
g.ViewHandlers += delegate(View v) {
    Console.Title = "myGroup members: "+v.members;
};
g.Handlers[UPDATE] += delegate(string s, double v) {
    Values[s] = v;
};
g.Handlers[LOOKUP] += delegate(string s) {
    Reply(Values[s]);
};
g.Join();

g.Send(UPDATE, "Harry", 20.75);

List<double> resultlist = new List<double>;
nr = g.Query(LOOKUP, ALL, "Harry", EOL, resultlist);
```

- First sets up group
- Join makes this entity a member. State transfer isn't shown
- **Then can multicast, query. Runtime callbacks to the "delegates" as events arrive**
- Easy to request security (g.SetSecure), persistence
- "Consistency" model dictates the ordering seen for event upcalls and the assumptions user can make

Isis² makes developer's life easier

15

```
Group g = new Group("myGroup");
g.ViewHandlers += delegate(View v) {
    Console.Title = "myGroup members: "+v.members;
};
g.Handlers[UPDATE] += delegate(string s, double v) {
    Values[s] = v;
};
g.Handlers[LOOKUP] += delegate(string s) {
    Reply(Values[s]);
};
g.Join();

g.Send(UPDATE, "Harry", 20.75);

List<double> resultlist = new List<double>;
nr = g.Query(LOOKUP, ALL, "Harry", EOL, resultlist);
```

- First sets up group
- Join makes this entity a member. State transfer isn't shown
- Then can multicast, query. Runtime callbacks to the "delegates" as events arrive
- **Easy to request security (g.SetSecure), persistence**
- **"Consistency" model dictates the ordering seen for event upcalls and the assumptions user can make**

Isis² makes developer's life easier

16

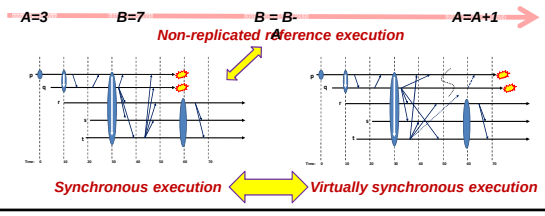
```
Group g = new Group("myGroup");
g.ViewHandlers += delegate(View v) {
    Console.Title = "myGroup members: "+v.members;
};
g.Handlers[UPDATE] += delegate(string s, double v) {
    Values[s] = v;
};
g.Handlers[LOOKUP] += delegate(string s) {
    Reply(Values[s]);
};
g.SetSecure();
g.Join();

g.Send(UPDATE, "Harry", 20.75);

List<double> resultlist = new List<double>;
nr = g.Query(LOOKUP, ALL, "Harry", EOL, resultlist);
```

- First sets up group
- Join makes this entity a member. State transfer isn't shown
- Then can multicast, query. Runtime callbacks to the "delegates" as events arrive
- **Easy to request security** (g.SetSecure), persistence
- "Consistency" model dictates the ordering seen for event upcalls and the assumptions user can make

Consitency model: Virtual synchrony meets Paxos (and they live happily ever after…)

17

- **Virtual synchrony is a "consistency" model:**
  - ***Membership epochs:** begin when a new configuration is installed and reported by delivery of a new "view" and associated state*
  - ***Protocols run "during" a single epoch:** rather than overcome failure, we reconfigure when a failure occurs*



How wo

18

We build the group as the system runs. Each participant just adds itself.

The leader monitors membership. This particular version doesn't handle failures but the "full" version is easy.

We can trust the membership. Even failure notifications reflect a system-wide consensus.

```
Group g = new G
g.ViewHandle      = dele
    IMPORT "db-replica:"+v.GetMyRank();
};
g.Handlers[UPDATE] += delegate(string s, double v)
{
    START TRANSACTION;
    UPDATE salary = v WHERE SET
    COMMIT;
};
...

g.SafeSend(UPDATE, "Harry", "8
```

1. **Modify the view handler to bind to the appropriate replicate (db-replica:0 ...)**

Paxos guarantees agreement on message set, the order

This code requires that mySQL is deterministic and that the serialization order won't be changed by QUERY operations (read-only, but they might get locks). As it happens, those assumptions are valid.

Drilling down: Is this correct?

- This question combines several kinds of reasoning
  - Is Isis^2 itself correct?
    - E.g.: Are the conditions assumed by the system satisfied by the runtime setting, does it use correct protocols, was the system implemented correctly and properly tested, etc.
  - Is this application using the system correctly?
  - Does the end-user have the same notion of correctness that the system achieves?
    - For example, if the end-user wants to be sure that updates to the database are ordered and persistent, will this snippet of code actually achieve those goals?
- How can we exploit formal tools and languages to help us answer these kinds of questions?

The system uses its own model

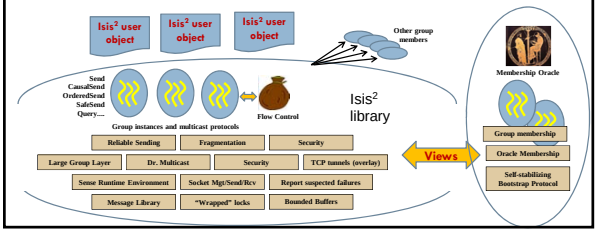
- Isis^2 implementation makes extensive use of model!
- Model-driven reasoning was key to building the many protocols and mechanisms that the system packages on behalf of its users
- In some sense the model is a cornerstone on which we've constructed a castle.

Paradox

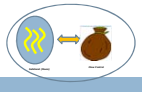
- Robbert van Renesse has an Erlang implementation of Paxos in this model: about 60 lines of very elegant code
  - With NuPRL can prove such implementations correct!
- In contrast, Isis^2 has about 10,000 semicolons in C#
  - And my code is far more complex
  - Why not just build Isis^2 from Robbert's little module?

It takes a "community"

- Formal methods tempt us to reason about a single instance of a single protocol at a time:
  - "Paxos with *n* members = { x, y, z, ... } and α acceptors..."
- Yet real systems are complex and concurrent with many interacting component parts that must operate in concert

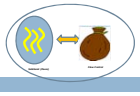


Consider flow control

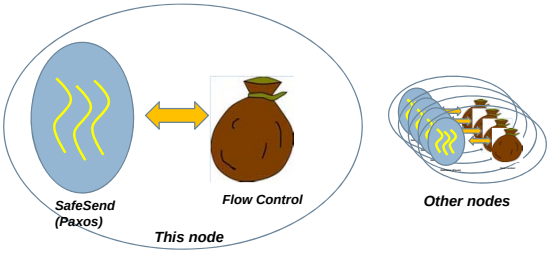


- Consider SafeSend (Paxos) within Isis^2
  - Basic protocol looks very elegant
  - Not so different from Robbert's 60 lines of Erlang
- But pragmatic details clutter this elegant solution
  - E.g.: Need "permission to send" from flow-control module
  - ... later tell flow-control that we've finished
- Flow control is needed to prevent overload
  - Illustrates a sense in which Paxos is "underspecified"

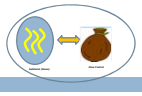
Pictoral representation



- "Paxos" state depends on "flow control state"
- Modules are concurrent. "State" spans whole group



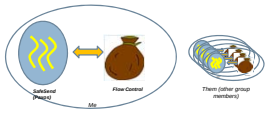
… flow control isn't local



- One often thinks of flow control as if the task is a local one: "don't send if my backlog is large"
- But actual requirement turns out to be distributed
  - "Don't send if the *system as a whole* is congested"
    - Permission to initiate a SafeSend obtains a "token" representing a unit of backlog at this process
    - Completed SafeSend must return the token
    - Flow Control module tracks backlog states of full set of group members, hence needs a rule for reporting state via multicast
    - Must also monitor group membership and unblock senders if a failure "frees" enough backlog to enable senders to resume
  - Thus Flow Control is a non-trivial distributed protocol!

This creates a new challenge

- Previously, could have proved Paxos safe+live in the virtual synchrony model
  - Virtual synchrony views play the role of a failure detector (an eventually strong one, in the sense of ◊ S)
  - Paxos lives in a simpler world and can be proved fully correct
- But now we see that Paxos would be "dependent" upon the flow control module, and vice versa!
  - Paxos needs permission to send
  - Flow control needs to track protocols in progress
  - Group members need to track each-other's states



Paxos + Flow Control correctness?

- Flow control imposed only when a protocol starts
  - Waiting for flow control induces a partial dependency ordering
  - If prior protocols are live, some waiting protocol will eventually have a chance to run
  - Fairness requires further mechanisms…

It isn't quite so simple: Delay Tolerance

- Recall that Isis^2 targets cloud-scale settings
  - Hence aggressively scaled, must "ride out" scheduling delays, long message latencies, elasticity events
  - Most work on DTNs focuses on progress "despite" delays
  - But in Isis^2 if some nodes get far ahead of other nodes, the flow-control module we've just discussed kicks in! This defeats DTN logic
- Given this mix of needs, which the best 2PC implementation?
  - One leader, *n* members
  - Hierarchical (tree)
  - Tree of rings (Ostrowski: QSM)
  - Hypothetical: Self-stabilization or gossip "emulation" of 2PC

- … And whichever we favor also needs to lend itself to an implementation we can prove correct!

Lessons one learns… and challenges

- Formal models are powerful conceptual tools
  - Impossible to build a system like Isis^2 without them
  - And Isis^2 in turn enables high-assurance applications
- Yet our science of formal methods remains too narrow in its focus
  - Teaches us how to reason about a single protocol
  - But also need to think about communities of protocols, concurrency everywhere, cross-process dependencies

What about the code we saw earlier?

- In fact, combining Isis^2 (or Paxos) with MySQL this way involves all sorts of hidden assumptions and brings all sorts of implied obligations
- Many researchers who use these sorts of techniques do so without understanding those issues
  - Hence surprisingly many prominent research papers are flawed, or at least don't tell the whole story!
  - In CS6410 we'll try to drill down to those insights

Reminder: MySQL replicated with Isis²

31

```
Group g = new Group("myGroup");
g.ViewHandlers += delegate(View v) {
   IMPORT "db-replica:"+v.GetMyRank();
};
g.Handlers[UPDATE] += delegate(string s, double v)
{
   START TRANSACTION;
   UPDATE salary = v WHERE SET name=s;
   COMMIT;
};
...

g.SafeSend(UPDATE, "Harry", "85,000");
```

1. **Modify the view handler to bind to the appropriate replicate (db-replica:0, ...)**
2. **Apply updates in the order received**
3. **Use the Isis² implementation of Paxos: SafeSend**

What about the code we saw earlier?

- In fact, combining Isis² (or Paxos) with MySQL this way involves all sorts of hidden assumptions and brings all sorts of implied obligations
- Many researchers who use these sorts of techniques do so without understanding those issues
  - Hence surprisingly many prominent research papers are flawed, or at least don't tell the whole story!
  - In CS6410 we'll try to drill down to those insights

Issues to ponder:

- Assumes that MySQL is deterministic when replicated. Is this true? **[Yes]**
- There are failure cases in which Isis² could achieve its platform-level atomicity guarantee and yet the MySQL updates might be partially complete, because they occur "after" multicast delivery. The code I showed was missing the needed cleanup. **[Must add a log-replay]**
- For performance reasons, we would want to analyze the pattern of updates and optimize for what arises in practice. **[ok...]**
- Isis² sometimes requires the user to call "g.Flush()" but this code didn't do so. What is g.Flush, when is is required, and was this code correct or incorrect to omit that particular action? **[SafeSend doesn't need Flush]**

The challenge?

- Which road leads forward?
  1. Extend our formal execution model to cover all elements of the desired solution: a "formal system"
  2. Develop new formal tools for dealing with complexities of systems built as communities of models
  3. Explore completely new kinds of formal models that might let us step entirely out of the box

The challenge?

- Which road leads forward?
  1. Extend our formal execution model to cover all elements of the desired solution: a "formal system"

Doubtful:
- **The resulting formal model would be unwieldy**
- **Theorem proving obligations rise more than linearly in model size**

The challenge?

- Which road leads forward?
  1. Extend our formal execution model to cover all elements of the desired solution: a "formal system"
  2. Develop new formal tools for dealing with complexities of systems built as communities of models

Possible, but hard:
- **Need to abstract behaviors of these complex "modules"**
- **On the other hand, this is how one debugs platforms like Isis²**

The challenge?

- Which road leads forward?
  1. Extend our formal execution model to cover all elements of the desired solution: a "formal system"
  2. Develop new formal tools for dealing with complexities of systems built as communities of models
  3. Explore completely new kinds of formal models that might let us step entirely out of the box

Intriguing:
- **All of this was predicated on a style of deterministic, agreement-based model**
- **Could self-stabilizing protocols be composed in ways that permit us to tackle equally complex applications but in an inherently simpler manner?**

CS6410 versus just-read-papers

- A paper on Isis^2 might just brag about how great it is, how well it scales, etc
- Reality is often complex and reflects complex tensions and decisions that force compromises
- In CS6410 our goal is to be honest about systems: see what the authors had to say, but think outside of the box they were in when they wrote the papers

Why take this course

- Learn about systems abstractions, principles, and artifacts that have had lasting value,
- Understand attributes of systems research that is likely to have impact,
- Become comfortable navigating the literature in this field,
- Learn to present papers in a classroom setting
- Gain experience in thinking critically and analytically about systems research, and
- Acquire the background needed to work on research problems currently under study at Cornell and elsewhere.

Who is the course "for"?

- Most of our CS6410 students are either
  - PhD students (but many are from non-CS fields, such as ECE, CAM, IS, etc)
  - Undergraduates seriously considering a PhD
- A small subset are MEng students
  - Some MEng students are ok pretending to be PhD students and have the needed talent and background
  - MEng students not fitting this profile won't get permission to take the course
  - CS5410 was created precisely to cover this kind of material but with more of an MEng focus and style

Why take this course

- CS 6410 is one wat to satisfy the systems breadth requirement (CS64xx in the "rubric")

		Research Styles		
		Theoretical	Systems	Applied
Areas	Algorithms/Theory	68xx		
	AI	676x		67xx except 676x
	Systems		632x, 64xx	
	PL	6110		
	Sci. Comp. and Apps			62xx, 65xx, 66xx

Required background

- A desire to learn about the research frontier in systems: cutting edge questions that may be somewhat divorced from practice, at least at first
  - In fact systems people view "not real" stuff as being "not research", so most ideas are very real
    - But not every real thing turns out to be a success in industry and many great ideas are never adopted
    - CS5410 tends to be more focused on practical, useful insights into how industry really does things.
- We expect you to already have very solid background in systems: architecture, operating systems, perhaps database or other storage systems

Details

How class operates and other practical stuff

Details

- Instructor: Ken Birman
  - ken@cs.cornell.edu
  - Office Location: 4119B Upson
- TA: None assigned, but Zhiyuan Teo and Qi Huang are willing to help out in small ways
  - zt27@cs.cornell.edu and qhuang@cs.cornell.edu
- Lectures:
  - CS 6410: Tu, Th: 10:10 – 11:25 PM, 140 Bard Hall

Course Help

- Course staff, office hours, announcements, etc:
  - http://www.cs.cornell.edu/courses/cs6410/2012fa
- Please look at the course syllabus: the list of papers is central to the whole concept of this class
- Research project ideas are also listed there

CS 6410: Overview

- Prerequisite:
  - Mastery of CS3410, CS 4410 material
    - Fundamentals of computer architecture and OS design
    - How parts of the OS are structured
    - What algorithms are commonly used
    - What are the mechanisms and policies used
  - Some insights into storage systems, database systems "helpful"
  - Some exposure to networks, web, basic security ideas like public keys
- Class Structure
  - Papers Readings (whole semester)
  - Paper Presentations (whole semester)
  - Labs (first 1/8)
  - Research Project (second 7/8)

CS 6410: Topics:

- Operating Systems
  - Core concepts, multicore, virtualization, uses of VMs, other kinds of "containment", fighting worms/viruses.
- Cloud-scale stuff
  - Storage systems for big data, Internet trends, OpenFlow
- Foundational theory
  - Models of distributed computing, state machine replication and atomicity, Byzantine Agreement.
  - Impact of social networks, P2P models, Self-Stabilization

CS 6410: Paper Readings

- Required reading: 2 or 3 papers
  - Reflecting contrasting approaches, competition, criticism,…
  - Papers pulled from, best journals and conferences
    - TOCS, SOSP, OSDI, …
  - 26 lectures, 54 (required) papers + 50 or so "recommended"!
- Read papers before each class and bring notes
  - takes ~2 to 3 hrs per paper, write notes and questions
  - Some papers may take 4 hours to understand
- Write a review and turn in at least one hour before class
  - Turn on online via Course Management System (CMS)
  - No late reviews will be accepted

CS 6410: Writing Reviews

- Each student is required to prepare notes on each paper before class and to bring them to class for use in discussion.
- Your notes should list assumptions, innovative contributions and criticisms.
  - Every paper in the reading list has at least one major weakness.
  - Don't channel the authors: your job is to see the bigger questions!
- Turn paper reviews in online before class via CMS
  - Be succinct—One paragraph per paper
    - Short summary of paper (two or three sentences)
    - Two to three strengths/contributions
    - and at least one weaknesses
  - One paragraph to compare/contrast papers
  - In all, turn in two to three paragraphs

CS 6410: Paper Presentations

- Each person will present a paper one or two times, depending on the stabl class size
  - Read and understand both required and suggested papers
  - Learning to present a paper is a big part of the job!
- Two and a half weeks ahead of time
  - Meet with professor to agree on ideas to focus on
- One and a half weeks ahead of time
  - Have presentation prepared and show slides or "chalk talk" to professor
- One week ahead of time
  - Final review / do a number of dry-runs

CS 6410: Class Format

- 45-50 minutes presentation,
- 30 minutes discussion/brainstorming.
  - In that order, or mixed.
- All students are required to participate!
- Counts in final grading.

CS 6410: Research Project

- One major project per person
  - Or two persons for a very major project
- Initial proposal of project topic – due mid-September
- Survey of area (related works)–due begin of October
- Midterm draft paper – due begin of November
- Peer reviews—due a week later
- Final demo/presentation–due begin of December
- Final project report – due a week later

CS 6410: Project Suggestions

- Better system support for cloud computing.
- Operating systems that dynamically shift loads and reduce the risk of node overload.
- Extending enterprise VLAN technology into the Internet.
- Security policy for complex large-scale applications.
- Advances in mobile computing.
- Side-by-side "supernetworks".
- Next generation storage systems for the cloud.
- DDoS repelling cloud service.
- How could social networking systems make better use of P2P technology?
- How should self-stabilizing protocols be "integrated" with more deterministic ones?

Important Project Deadlines

9/13	Submit your topic of interest proposal
9/27	Submit 2-3 pages survey on topic
(Oct)	Discuss project topic with me
11/1	Midterm draft paper of project
11/29	Final demo/presentation of project
12/6	Final paper on project

CS 6410: Grading

- Class Participation ~ 40%
  - lead presentation, reading papers, write reviews, participation in class discussion
- Project ~ 50%
  - Proposal, survey, draft, peer review, final demo/paper
- Subjective ~ 10%
- This is a rough guide

Academic Integrity

- Submitted work should be your own
- Acceptable collaboration:
  - Clarify problem, C syntax doubts, debugging strategy
  - You may use any idea from any other person or group in the class or out, provided you clearly state what you have borrowed and from whom.
  - If you do not provide a citation (i.e. you turn other people's work in as your own) that is cheating.
- Dishonesty has no place in any community
  - May NOT be in possession of someone else's homework/project
  - May NOT copy code from another group
  - May NOT copy, collaborate or share homework/assignments
  - University Academic Integrity rules are the general guidelines
- Penalty can be as severe as an 'F' in CS 6410

Stress, Health and Wellness

- Need to pace yourself to manage stress
  - Need regular sleep, eating, and exercising
- Don't miss class... but....
- Do not come to class sick (with the flu)!
  - Email me ahead of time that you are not feeling well
  - People not usually sick more than once in a semester

Before Next time

- Rank-order 2 papers to present (first and second half)
- Read first papers below and write review
  - End-to-end arguments in system design, J.H. Saltzer, D.P. Reed, D.D. Clark. ACM Transactions on Computer Systems Volume 2, Issue 4 (November 1984), pages 277--288.
    http://portal.acm.org/citation.cfm?id=357402
  - Hints for computer system design, B. Lampson. Proceedings of the Ninth ACM Symposium on Operating Systems Principles (Bretton Woods, New Hampshire, United States) 1983, pages 33--48.
    http://portal.acm.org/citation.cfm?id=806614
- Check website for updated schedule