THE INTERNET OF THINGS: A BIG DEAL

Smart power grid…
Smart highways to guide smart cars…
Smart homes

Huge market

Scale demands a cloud with special properties
A “SMART” POWER GRID?

Today’s grid uses mostly 1980’s technology

Human-controlled… robust, but struggling with renewable energy.

Core idea: Machine learning could bring transformative flexibility
FORMS OF MICROGENERATION

Solar panel on the rooftop

Battery in the wall

Heater or A/C unit that we can activate early or delay

➢ Think of “delayed power use” as a new form of power: “schedulable” demand!
SMART METERS

Like attaching an iPhone to the wall in the utility closet

Knows what’s going on, regulates power consumption for heavy power-use devices like A/C, heater, etc.

Ideally, the “endpoint partner” for the more centralized optimization codes that work to balance supply and demand
The intelligence lives in the cloud. The external infrastructure is mostly dumb sensors and actuators.
CAN THE CLOUD DO REAL-TIME?

Internet of Things, Smart Grid / Buildings / Cars, . . .

Shared requirement:

- We want a system that can carry out some form of continuous monitoring, or continuous control.
- It will need to be robust despite “cloudy weather” and offer quick response, often with some form of consistency or fault-tolerance requirement added to the mix.
The shard members keep logs of values received indexed by time.

Due to network delay, not all have the same data at the same time.

We send identical data on all connections. Replication overcomes the network delays.

Consistency: Do the components use correct “wiring diagram”?

Even if one link is slow, the others might be ok.
FORMS OF CONSISTENCY

Cloud elements exposing replicated data should show the same system state, modulo things changing right as you look

Applications are like operators: same need

But what if the cloud computing platform causes inconsistency?

- We think of these systems as if they were single programs
- Actually they are spread over many nodes in the cloud
FORMS OF CONSISTENCY

One issue is real-time consistency
- After an event occurs, it should be rapidly reported in the cloud
- And anyone sharing the platform should soon see it

Another issue is replication consistency
- Replicate for fault-tolerance and scale
- But all replicas should have identical contents (within a brief delay)
CORNELL RESEARCH AGENDA

Build on today’s cloud, but bring super-fast consistency solutions to bear on these needs

Outcome: a nimble real-time cloud with consistency

Best of all: not only does our stuff work better, it is way faster too
FREEZE-FRAME FS

Real-time file system for secure, strongly consistent data capture
- Normal file system but understands real-time timestamps
- Offers optimal temporal precision plus Chandy-Lamport consistency

Incredibly high speed
- Leverages RDMA for network line-speed data transfers
- NVRAM (SSD or RAID disks) for storage persistency
We simulated a wave and sampled it. Like taking photos of individual grid cells (squares). Then streamed the data to our cloud-hosted historian, one stream per sensor. Then reconstructed the wave from the files and created these gif animations.
HOW DOES FFFS WORK?

FFFS CAPTURES DATA PLUS TIME

INTERNALLY, A SCALABLE, FAULT-TOLERANT MEMORY-MAPPED DATA INDEX
HOW DOES FFFS WORK?

When you issue a read, there are two cases:

- **Read “now”** (standard POSIX API and behavior)
- **Read @ time**: FFFS finds and returns the data applicable at the time you requested:
  - Writes: time extracted from data records (plug-in tells us how)
  - ... or we can use “platform time” when the write occurred
  - Reads: POSIX applications read from a directory named by time
  - ... or can use a non-POSIX API with time as an extra parameter.
TIME MATTERS!

For real-time applications, we’ll need cloud services that understand time

Why it matters: if you run deep-learning on temporally fuzzy data the algorithm will suffer from really bad input

With Freeze Frame FS, we can start to run powerful file-oriented analytics that understand data, and that keep up in real-time
WHAT ABOUT RAW SPEED? FOR THAT…

**FFFS:** Speed from memory-mapped data, RDMA transfers

**RDMC:** Reliable data replication at insane speeds

**SST:** A new simple form of shared memory for rack-scale systems

**Derecho:** \{RDMC+SST\} used to create ultra-reliable replicated data with virtual synchrony groups, multicast, Paxos

… and it all works, open source. Download it now if you like
IDEA: BUILD AN RDMA MULTICAST

RDMA: Direct **zero copy** transfers from user memory to user memory

On 40 Gb/s network TCP/IP hits 250Mb/s... RDMA reaches 36Gb/s

- This is a comparison with TCP on IP, not TCP optimized for RDMA.

Like TCP, RDMA is reliable: if something goes wrong, the sender or receiver gets an exception. This only happens of one end crashes

- RDMA also has an unreliable built-in multicast, but we don’t use it
TURN UNICAST RDMA INTO **MULTICAST**

**RDMC**: RDMA protocol used in a multicast

![Diagram of Binomial Tree, Binomial Pipeline, and Final Step]
HOW FAST IS RDMC?

Up to 16 copies we peg the optical network...

Then slow degradation due to link contention (idea: topologically aware solution could avoid this)

RDMC should eventually scale to thousands of replicas
HOW FAST IS RDMC?

This shows RDMC messages per second for very small messages

Peak rate is about 120K/s

Using Mellanox 100Gbps RDMA on ROCE (fast Ethernet)
20Gb/s = 2.5GB/s.... 100Gb/s = 12.5GB/s
HOW FAST IS RDMC?

This graphs bandwidth for various message sizes.

RDMC can use the full RDMA bandwidth (12.5GB/s).

This is 3x faster than you can copy memory-to-memory.

Using Mellanox 100Gbps RDMA on ROCE (fast Ethernet)

20Gb/s = 2.5GB/s.... 100Gb/s = 12.5GB/s
A SECOND EXAMPLE: SST

A simple shared table implemented with RDMA

Each node owns one row. If you update “your” row, the data is pushed to other nodes

SST also support events: “on (predicate) do { function }”

- Current version: close to 1 million events per second with 8 nodes
- Same memory model as for “volatile” variables in C++
Derecho is a group communication infrastructure

Programs can attach to groups (“join”) them

Data is replicated by doing a group send operation
A PROCESS JOINS A GROUP

At first, P is just a normal program, with purely local private variables.
P still has its own private variables, but now it is able to keep them aligned with track the versions at Q, R and S.

S starts by creating an endpoint object, attaching upcall handlers to (later) react to events like membership changes and multicasts reporting updates.

Once the group endpoint is properly configured, S issues a “join” request. Vsync checks to see if the group already exists. If not, p can create a new instance, but in this case, the group is already active.
A PROCESS AS A GROUP MEMBER

All members see the same “view” of the group, and see the updates (multicasts) in the identical order.
A PROCESS FAILS

All members see the same “view” of the group, and see the updates (multicasts) in the identical order.
**CONSISTENCY MODEL: VIRTUAL SYNCHRONY + PAXOS**

- **Virtually synchronous run** is indistinguishable from behavior of a non-replicated object that saw the same updates (state machine replication).
- **Persistent virtually synchronous runs** are the same as Paxos.

Excerpt:

- Virtually synchronous run is indistinguishable from behavior of a non-replicated object that saw the same updates (state machine replication).
- Persistent virtually synchronous runs are the same as Paxos.
**DERECHO = RDMC + SST**

Derecho group with members \{A, B, C\} in which C is receive-only

DNS: myGroup.somewhere.org | 123.45.67.123:6543 (Leader: A)

\[ V_3 = \{ A, B, C \} \]

DNS record and current view, showing senders

<table>
<thead>
<tr>
<th>Suspected</th>
<th>Proposal</th>
<th>nCommit</th>
<th>Acked</th>
<th>nReceived</th>
<th>Wedged</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>4: -B</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
... INSANELY FAST REPLICATION

As a multicast: Making 256 replicas takes just twice the time of 1!

10,000x faster than older multicast libraries like Ken’s Vsync.

Write to SSD for Paxos properties. This yields scaling and performance that blows away standard Paxos and doubles what Microsoft’s record-setting Corfu system reports
DERECHO IS A MULTICAST AND PAXOS

As a multicast, moves data from DRAM to DRAM at insane speeds

- Reliable, totally ordered, can use the “view” to coordinate

As a storage solution, moves data to NVRAM (SSD disk)

- Now it has exactly the properties of Lamport’s Paxos protocol

Derecho.codeplex.com
SO... WHAT ABOUT CONTAINERS?

Container trend is the key to the game!

Virtualization is at odds with RDMA: very hard to securely share RDMA NICs in a virtualized setting with enterprise VLANs etc.

Containers strip away the virtualization layer and enable us to leverage these kinds of insanely fast data sharing, replication and retrieval technologies.
KEY TAKEAWAY?

The Internet of Things revolution is extremely promising for all of us.

Huge amounts of data will need to be captured, persisted. Many obvious roles for non-volatile memory.

Containers will win because of their lower overheads and ability to leverage RDMA and NVRAM hardware more effectively.
The Internet of Things is coming… and will bring extreme scale.

We need to learn to move big data in real-time with strong consistency properties.

Freeze Frame FS and Derecho are proof that if we leverage RDMA and SSD hardware, it can be done.