Motivation: We started looking at real-time cloud control of IoT systems (specific use case: Smart Power Grid).

... ended up building GridCloud, and ISO NE, NYPA and NY ISO are our users. Such uses need real-time responsiveness, ultra-fast data replication, strong consistency. Today’s cloud doesn’t do those things.

So, we decided to build Derecho: World’s fastest platform component for replication and coordination.

Now hoping that we can lay groundwork for an open-source Derecho community that might want to use this technology, and help improve it
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
IOT: PATHWAY TO 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
IOT DIMENSION: 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

In-house control center for all these fairly dumb devices

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 cloud-based optimization that works to balance supply and demand
BUT CAN TODAY’S CLOUD DO THIS?

Existing platforms have great real-time responsiveness only when serving requests at the edge, from what could be stale data.

Inconsistency is pervasive, CAP is a kind of strange folk religion.

Fault handling and dynamic reconfiguration can provoke waves of inconsistency and performance variability that may last for minutes. **Our goal: Fix these things, but build on standards.**
The intelligence lives in the cloud. The external infrastructure is mostly dumb sensors and actuators.
GRIDCLOUD 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)
First, we’ll just touch on high-level components

Then our deep dive will look closely at Derecho: Our new platform module (like an OS microkernel) that offers a one-stop shop for all our data replication, coordination and consistency requirements.

Derecho is where most of our real innovation resides.
Elements of GridCloud

Data collection network: Redundant TCP. Have also explored a fancier SDN-based solution we call IronStack.

Management framework: CM (“cloud make”) mimics Make but is parallel and maps control actions to dependency graphs.

Archival historical: Freeze Frame File System. I’ll say more about this in a second. Currently extending it with a Kafka API.

Common need: Run these elements on the cloud, but offer strongly consistent real-time data replication, coordination, fault tolerance
DEEP DIVES

We don’t have a huge amount of time

So look closely at one high-level element (Freeze Frame FS)

Then drop down in the OS communication layer and focus on Derecho, our platform for ultra-fast data replication
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.
IOT NEEDS CONSISTENT TIME!

Real-time applications need cloud services that understand time

- Deep-learning on temporally fuzzy data will give incorrect output
- Operators and systems using stale data will make poor decisions

With Freeze Frame FS, we can start to run powerful file-oriented analytics that understand data, and that keep up in real-time
HOW DOES FFFS WORK?

FFFS CAPTURES DATA PLUS TIME

INTERNALLY, KEEPS A SCALABLE, FAULT-TOLERANT MEMORY-MAPPED DATA SET INDEXED BY TIME
API SUPPORTS READING AND WRITING TEMPORAL DATA

- Read “now” (standard POSIX API and behavior). FFFS finds and returns the data applicable at the time you requested:
  - POSIX applications read from a directory named by desired time
  - … or can use a non-POSIX API with time as an extra parameter.

- Writes: time extracted from data records (plug-in tells us how)
  - … or we can use “platform time” when the write occurred
  - … or you can just tell us using a non-POSIX API
The NameNode is replicated for fault-tolerance and sharded for scalability. Membership consistency matters as does meta-data consistency.

The DataNodes are similarly replicated, and each is a shard. Management of this ensemble also poses high-speed coordination and fast-reaction challenges.
AND NOW... THE REALLY DEEP DIVE

We say goodbye to our application layer, although it contains many more interesting new technologies.

... and dive down into the world of RDMA communication
RDMA: Direct zero copy from source memory to destination memory

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

- This is a comparison with TCP on IP.
- Mellanox and Intel offer versions of TCP that use RDMA if available.

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

When RDMA hardware is available, this lets us blast past any file system you’ve ever seen!

- Our experiments run on Mellanox 100Gb/s routers
- We’ve worked with both ROCE (Ethernet) and Infiniband and got fantastic performance in both cases

When RDMA hardware isn’t available we just use TCP (SoftROCE) but even in this mode, run faster than standard cloud file systems
MORE CORNELL RDMA STUFF…

**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.
RDMC: AN RDMA MULTICAST

Binomial Tree

Binomial Pipeline

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?

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

Using Mellanox 100Gbps RDMA on ROCE (fast Ethernet)

20Gb/s = 2.5GB/s…. 100Gb/s = 12.5GB/s
HOW FAST IS RDMC?

For small messages, we use a special protocol we call the SST multicast.

With 1 sender, 250K msgs/s

If all members send at the same time, we reach nearly 2M msgs/s delivered to each member with latency of about .5us

Using Mellanox 100Gbps RDMA on ROCE (fast Ethernet)

\[
20\text{Gb/s} = 2.5\text{GB/s} \quad \text{...} \quad 100\text{Gb/s} = 12.5\text{GB/s}
\]
RDMC is like a torrent of data pouring down a flooded river. Reliable (unless something crashes; we’ll discuss that soon) but at a crazy, insane data rate.

But how can we offer stronger Paxos-style properties without losing all this awesome speed?

Key idea: Asynchronous “control” plane that uses monotonic properties to express goals, and runs out of band from the data
MONOTONIC PROPERTIES?

... just like the term sounds: “things that remain true, once they are established.”

- Example: if a counter is greater than 10, it remains greater than 10 even if incremented.
- Another example: if a replicated object has been safely delivered to all its destinations and persisted by them, and the ordering is agreed upon, in the Paxos model this is a “stable” or “monotonic” property. Once safe, always safe.
SO HERE’S THE IDEA…

Build a little tool to let us attach strong properties to our raging torrent of data

We call this the SST (shared state table). It uses RDMA too.

Then re-express class protocols as monotonic predicates using the SST for the variables (the shared state) needed to sense strong properties like Paxos safety (we have other use cases in mind too)
A **simple shared table** implemented with RDMA

Key idea: each element of a rack-scale system owns one row in the table. The row format is defined by a C++ 11 struct that has well known sizes, “plain old data”

If you update “your” row, the data is pushed to other nodes

Then we support events: “on (predicate) do { function }”

- Current version: close to 1 million events per second with 8 nodes
- Runs on one-sided RDMA writes (or reads, but writes are faster)
- Same memory model as for “volatile” variables in C++

<table>
<thead>
<tr>
<th>#</th>
<th>Load</th>
<th>Queue Backlog</th>
<th>Cache hit rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.4</td>
<td>12</td>
<td>17.6</td>
</tr>
<tr>
<td>2</td>
<td>3.1</td>
<td>3</td>
<td>79.9</td>
</tr>
<tr>
<td>3</td>
<td>7.8</td>
<td>23</td>
<td>4.6</td>
</tr>
</tbody>
</table>
Combining RDMC and SST: Derecho

Derecho is a group communication infrastructure.

Programs can attach to groups ("join") them.

Data is replicated by doing a group send operation.
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.

... Automatically transfers state ("sync" of S to P,Q,R)

Now S will receive new updates.
A PROCESS AS A GROUP MEMBER

All members see the same “view” of the group, and see the updates (multicasts) in the identical order.
All members see the same “view” of the group, and see the updates (multicasts) in the identical order.
CLOUD SERVICES OFTEN HAVE SUBGROUPS, AND THOSE ARE OFTEN “SHARDED” FOR SCALING

External clients use standard RESTful RPC through a load balancer

Multicasts used for cache invalidations, updates
DERECHO OFFERS ALL OF THIS

Extremely simple API focuses on

- Point to Point Send, RPC-style Query
- Multicast Send and Query

Yet we can cover all of these cases, and moreover, by working in C++ 17, obtain super-efficient marshalling, polymorphism
API SUMMARY: P2P SEND, P2P QUERY

```javascript
var outcome = g.P2PSend<MemCacheD@put>(who, “John Doe”, 22.7);
var result = g.P2PQuery<MemCacheD@put>(who, “Holly Hunter”);
```

class MemCacheD : public DerechoClass<MemCacheD>{
    HashSet<string, double> underlying_map;
    @EntryPoint void put(string s, double v){code...}
    @EntryPoint @ReadOnly double get(string s){code...}
    serializes(underlying_map);
    void newView(View& new_view) [[override]] {...}
    MemCacheD(group g):DerechoClass<MemCacheD>(g){code...}
};
...
```
Group<MemCacheD, CacheLayer, ...> newgroup{ [args_for_first], [args_for_second]};
...
```
API SUMMARY: GROUP MULTICAST/QUERY

```java
var g = Join<Paxos>("myGroup");

var outcome = g.OrderedSend<MemCacheD@put>("John Doe", 22.7);

for(var res: g.OrderedQuery<MemCachedD@get>("John Doe")) { code... }
```
Creating a single subgroup: a function specifies which parent members in a given view are members of the particular subgroup

\[
g.\text{designateSubgroup}\langle \text{LoadBalancer} \rangle \rightarrow \text{std::bitvec} \{ \text{return \{true,true,true,false\ldots \} } \}
\]

The sharded version specifies **two** lambdas: **one to compute the number of shards** and **one to designate membership for each shard**:

\[
g.\text{designateShardedSubgroup}\langle \text{CacheLayer} \rangle \rightarrow \text{std::bitvec} \{ \text{return \{true/false, \ldots \} } \}
\]
Cache Layer
Back-end Store
Multicasts used for cache invalidations, updates

Load balancer
External clients use standard RESTful RPC through a load balancer

... GENERATING THIS COMPLICATED SERVICE!
CONSISTENCY: A PERVERSIVE GUARANTEE

Derecho encourages what we refer to as “monotonic reasoning” Membership data (“views”) and group state is consistent across members, and these properties also extend to subgroups and sharded subgroups.

No surprises: the developer works with a step-by-step style of coding in which forced rollback never occurs (not even on failure) and in which knowledge is steadily accumulated during the run
Starting 10 years ago, Berkeley’s Eric Brewer posited a deep tradeoff between consistency, availability and partition tolerance.

Fundamentally: He argued that the cloud can’t afford consistency

With Derecho, consistency is inexpensive and scalable
**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
**IMPLEMENTATION: 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>4: -B</td>
<td>3</td>
<td>5</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>
**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
How Fast Is Derecho?

Using Mellanox 100Gbps RDMA on ROCE (fast Ethernet)

20Gb/s = 2.5GB/s.... 100Gb/s = 12.5GB/s

Derecho as a multicast protocol
In our Derecho formalism, the only difference is that with Paxos, we need to persist data (for example to SSD disk), whereas a multicast is identical but the data lives only in memory.

When configured to persist to SSD, the SSD itself is the bottleneck.

Our SSD has data-type dependent behavior.
Guarantees are exactly the same as for Paxos (Corfu)

Basically, a replicated persistent log on SSD storage units

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Speed of SSD (MB/s)</th>
<th>Derecho (MB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>YouTube video (mp4)</td>
<td>67.7645</td>
<td>63.2405</td>
</tr>
<tr>
<td>Random bits</td>
<td>72.1108</td>
<td>67.6758</td>
</tr>
<tr>
<td>Facebook web page</td>
<td>126.451</td>
<td>112.909</td>
</tr>
<tr>
<td>Repeating text data</td>
<td>457.55</td>
<td>300.398</td>
</tr>
<tr>
<td>All Zeros</td>
<td>471.104</td>
<td>292.986</td>
</tr>
</tbody>
</table>

4-member group, 4MB objects
We can also persist our data into FFFS

This yields a time-aware version of Derecho with persistence

Work is in progress... should be quite useful for IoT applications and in theory at least as fast as the Derecho Paxos
... INSANELY FAST REPLICATION

As a multicast: making 256 replicas is only 3x slower than one!

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

To SSD: Scale and performance that blows away standard Paxos and doubles what Microsoft’s record-setting Corfu system reports
CONTAINERS: KEY TO CLOUD RDMA

Cloud favors true virtualization, but RDMA has trouble with the resulting multi-level page tables.

Further, it is 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 RDMA in a shared environment.
WRAPUP: BIG PICTURE FOR IOT APPLICATIONS

API layer
- File system: FFFS
- Management: CM
- Pub/Sub: Kafka

Replication
- Derecho
- RDMC + SST

Containers
- Docker + Mesos or Linux or QNX

Hardware
- RDMA, SSD or next generation NV memory
  (SoftRoCE for backwards compatibility)
The Internet of Things revolution is extremely promising for all of us.

Huge amounts of data will need to be captured, persisted. Incredible demand for speed and real-time. **The cloud can do it!**

Containers will dominate virtualization as the choice for IoT platforms because of their lower overheads and ability to leverage RDMA and NVRAM hardware more effectively.
WHY NOT JOIN US?

The Internet of Things is coming... and will bring extreme scale

Google needs speed and consistency just as much as anyone else.

Derecho is an open-source project and with help, we could make it a universally adopted solution for the full space of replication in scalable settings. All of us win if a standard emerges!