HOW DO WE “PROGRAM” THE IoT CLOUD?

This is a very rapidly evolving and exciting question!

To give some context, let’s start with the same question as of 2005, to understand the answer today, but also the underlying reasons.

But the quick summary is: very few things work well, and we need to stick primarily to techniques that industry is prioritizing and supporting.
In the early period of cloud computing, there was a (mistaken) tendency to view the cloud like a very big distributed computing system.

So people learned to program on smaller clusters, then joined companies like Amazon, Yahoo!, eBay, and so forth, and took this knowledge along.

But in fact at cloud scale, those smaller techniques don’t work well!
To maximize concurrency, they started by spreading the work of building web responses over a set of side-by-side tier-one services.

The next challenge was to optimize the $\mu$-services. Remember Jim Gray’s paper!

- Some $\mu$-services can adopt CAP, because for them, “weak consistency is safe”.
- Some can use Paxos/SMR, like “one-shot transactions on a single shard.”
- Some use transactional database solutions, but on sharded data.
WHAT HAPPENS WHEN THEY GET IT WRONG?

**Convoy effect:** If some service runs slow, often a large amount of concurrent work “queues up” waiting for it and we lose all concurrency.

**Reboot storms:** When such a big overload arises that everything times out, crashes, and restarts.

**Inconsistency storms:** When a system that “normally” runs with good cached data suddenly finds that all cached data is extremely stale.
In lecture 1 we saw how companies like Amazon invented the \(\mu\)-service model. Now we can see that they had to go much further. They needed new families of \(\mu\)-services, and ways to teach people to build them.

Even with this, those \(\mu\)-services can become hot-spots. They used special hardware accelerators and design tools to program the new hardware.

This work wasn’t easy and required unusual expertise and luck.
Today’s cloud-scale solutions really work well!

But you really need to use them in the way the vendor anticipated. You generally download a “story book” that describes some end-user need and how the company helped solve it, and comes with sample code.

The code mostly glues together existing services. Then you download that code and customize it to transform it into your own solution, for your case.
CAN YOU FULLY CUSTOMIZE THE FIRST TIER?

In the 2006-2015 period or so, we saw increasingly sophisticated products emerge to help automate the creation of web pages.

So often, the first tier is just one of these vendor-supplied solutions and you use some form of special design tool to tell it what you are hoping to do.

For cases where this isn’t a good solution, we use a “hybrid cloud” in which applications inside the client company issue cloud requests directly.
HYBRID CLOUD (WEB SERVICES MODEL)

Basically, the cloud vendor offers cloud-hosted services via APIs you can call via RPC from a networked application. This runs on HTTPS.

So you can extend an application running inside your company to make requests to services up in the cloud, with good security.

Then any style of application development your company uses becomes cloud-enabled. But it still requires a certain level of expertise.
INTERNET OF THINGS BROKE THE MODEL!

With IoT, we encounter a slew of new issues.

➢ Customers lack a high quality way to securely manage huge numbers of IoT devices, which are often “dumb” sensors.

➢ These devices need to be actively managed, like to update their firmware each time a patch is issued, to protect them against hackers, ...

➢ There are thousands of devices and each has its own special, vendor-defined options for remote control.

➢ They often need real-time responses, large machine-learned knowledge bases that are frequently updated, fault-tolerance and consistency.
This is Ken’s SLR camera. What “events” can it generate?

➤ You don’t normally think of a camera as a device…. Now we want to imagine that this camera is being used as a cloud IoT peripheral with a stable source of power. Some possible events:

<table>
<thead>
<tr>
<th>Event</th>
<th>Meaning</th>
<th>Issues</th>
</tr>
</thead>
<tbody>
<tr>
<td>On/Off/Idle</td>
<td>Power mode</td>
<td>When turned on, must authenticate</td>
</tr>
<tr>
<td>NewIMG</td>
<td>Took a photo</td>
<td>Should we download it? How fast is the link?</td>
</tr>
<tr>
<td>StorageWarning</td>
<td>Low on space</td>
<td>Which images to delete</td>
</tr>
<tr>
<td>FocusWarning</td>
<td>Dirty sensor</td>
<td>Something is preventing auto-focus from working.</td>
</tr>
</tbody>
</table>
AN IOT EVENT WOULD...

Come from some specific device, securely attached to the cloud and with a clearly defined owner who bought it and services it.

Have a device type and an event type and “tags” defining event metadata, which is a term that just means that the event might talk about data but not include the raw data.

Why not include the data in every event?

- Typical photo might be 3MB in size. A video could be 1GB or more.
- But sensors rarely have ultra-fast connections.
IN AN IOT WORLD…

A typical end-user application (think of a hospital, or a small city, or a smart highway, or a smart farm) might have thousands or tens of thousands of smart sensors, of many kinds.

The devices would generally not be very smart, but each has its own superpowers, like on-camera storage for images, or image compression.

Different vendors/models: many “user manuals”

We have limited battery lifetimes and bandwidth to contend with, and perhaps can’t download every image or video.

Some data may be more valuable than others.
You can’t run code right on the sensor itself: they aren’t very smart.

You could build a specialized solution for each class of sensor, but this wouldn’t be cost-effective

So companies wanting to be big players in IoT have begun to invent a new IoT-oriented first tier.
EVENT-DRIVEN IOT MODEL

So… we securely bind Ken’s camera to our cloud, and register it under Ken’s account (he’ll be billed for the resources used). We use Azure IoT Hub.

Now the camera is accessible and we get events, and can send it events too:

- Cloud-to-camera: Turn on camera. Take one photo per second.
- Camera-to-cloud: PowerUp. {Status=success, 13GB free space}
- Camera-to-cloud: Photo {UID=IMG-2546.jpg, GPS=42.44940, -76.48280, …}
- Cloud-to-camera: DownloadPhoto {UID=IMG-2546.jpg, Quality=ThumbNail}
- Cloud-to-camera: DeletePhoto {UID=IMG-2546.jpg}
AZURE’S EVENT MODEL

First stage of filtering might handle, discard or transform the event

This is the normal Azure cloud, but extended to manage IoT devices in a smart way

HTTP://WWW.CS.CORNELL.EDU/COURSES/CS5412/2019SP
FUNCTION SERVERS

A very simplified way to create event-driven first-tier solutions.

- In Amazon, called Amazon Lambda
- Microsoft calls it the Azure Function Server

The server is an elastic set of machines that host very inexpensive containers running Linux or some other operating system of your choice.

You provide simple programs that get triggered by events.
FUNCTION MODEL

The idea, then, is to write code to handle these functions, for “both sides” of the connection.

Since the device itself is probably quite specialized and “dumb”, one side would be a kind of device driver that knows how to talk to it.

Then the cloud would be the other side of the connection.
HOW THE EVENT IS PASSED TO THE FUNCTION

When an event occurs, a new instance of the event handling function you registered will be launched in a “clean” state.

The event itself is available either as program arguments, or via an API.

You can also register a shell script if you wish.
ONE FUNCTION CAN HANDLE MULTIPLE EVENTS

This is a convenient way to avoid having to register hundreds of functions.

But the handler for any single event is normally very simple and short.

In Azure, the Visual Studio 2017 IDE can create a skeleton for you in C# or other .NET languages, with all the needed logic “sketched out”.

Then you would just extend the skeleton with specific logic of your own.
EXAMPLE LOGIC?

Your function could be passed a thumbnail photo, then use a photo-analysis μ-service to decide whether the photo

- Has any “content”. A farm photo with no animals might not be useful.
- Is of “interest”. Here, you could use a pre-trained machine learning classifier that has learned which kinds of photos interest you.

Then if the photo seems to be interesting, you could download the full version. If not, you might still leave the dull ones on the camera just in case.
FANCIER CASE: POINT, FOCUS AND SHOOT

Suppose that some event occurs: “Animal motion detected”. This will cause us to swivel the camera. When the camera is pointed towards the location, we should focus. When the focus converges, we shoot a photo. Now the thumbnail is sent to the server. If the photo is considered interesting, we’ll download it.
THIS IS A FORM OF STATE MACHINE!

Idle
- Handle various mundane events
- Motion event {args...}
  - Movement sensed! Point the camera
  - Send swivel command to Camera {args...}

Swivel
- Camera movement done {args...}
- Pointed! Focus the camera
  - Send focus command to Camera {args...}

Focus
- Focus operation done {args...}
- Focused! Take a photo
  - Send photo command to Camera {args...}

Take Photo
- Captured photo {info, }
BUILDING A STATE MACHINE WITH FUNCTIONS.

A function is “stateless”, which is a confusing idea for many people.

- Keep in mind: a function is simply an event-triggered program.
- It can access Azure μ-services (plus you can add extra μ-services).
- And so it can do persistent updates to the state in those μ-services.

To implement a state machine, even a simple one, we would have to keep the state itself external to the function: load it, then update it, then store it.
An atomicity issue arises if concurrent events trigger two functions that try to make conflicting updates to the state machine state.

If we solve this by adding locks to the key-value store, Jim Gray’s scalability warning applies!

Solution? We use a kind of “atomic” conditional key-value put.
STATE UPDATES

Current State: Data in the (key,value) store can hold any information you like

Version=17

New Event

Function launched to handle it

Updated state replaces prior state ("Replace state version 17 with state version 18")

Triggered action (issued after successful state update)
BUT YOU CAN ALSO SEE THAT THIS IS HARD

You wouldn’t encode long sequences of complex logic this way.

So a function service has handlers limited to very simple direct actions, or very simple sequences.

Anything more complex needs to be implemented in a μ-service
HOW WILL YOU KNOW WHICH TO USE?

Draw a diagram of the state machine you’ll need.

If it only has a very small sequences of simple steps, implementing them purely with functions and saving state in a $\mu$-service will be fine.

But if the state machine seems at all complex, you’ll probably need to either build your own $\mu$-service, or build a new $\mu$-service that leverages existing ones but still “holds” the longer-term stateful interactions.
EXISTING $\mu$-SERVICES FUNCTIONS OFTEN USE

Message bus (pure notification) and queues (stored until deleted)
File systems, key-value storage (with conditional updates!)
Services for compressing big objects, photo segmentation & tagging
Deduplication services ("did I already know this?")
Blockchains (like a tamper-proof audit trail, append-only)

…. Easy to dream up more of them specialized for particular needs, but they might not be easy to implement!
BUILDING NEW $\mu$-SERVICES

You would use a tool, like Cornell’s Derecho software library.

Derecho is a powerful infrastructure and we will learn much more about it soon. It lets you build complex self-managed services to run for long periods of time and self-heal if failures or other disruptions occur.

Many companies will probably $\mu$-Service development kits over time.
HOW FLEXIBLE IS THIS PARADIGM?

The claim is that it can cover a vast range of use cases!

One puzzle is to decide what belongs in a μ-service and what belongs in a function, since this “trick” of saving state allows the functions themselves to do fairly elaborate things.

The general model is to keep functions very simple and fast and put complex logic into the μ-service layer (adding new ones, if needed)
“The Internet of Things is transforming the everyday physical objects that surround us into an ecosystem of information that will enrich our lives. From refrigerators to parking spaces to houses, the Internet of Things is bringing more and more things into the digital fold every day, which will likely make the Internet of Things a **multi-trillion dollar industry** in the near future.” — Price-Waterhouse-Cooper report
FARMING IS JUST ONE OF MANY “MARKETS”

- Smart food delivery “pipeline” from farm to table.
- Smart homes, and cities. Smart highways and traffic light patterns.
- Smart “apps” for this smart world (like Elder Care, or Healthy Living)
- Smart power grid.
- Smart corporate campuses that adapt heating, lighting, humidity
- Smart protective systems to crack down on theft and other crimes
Azure IoT is a powerful infrastructure but to tackle these smart applications, you still need to do a fair amount of coding.

Function servers offer a great way to handle high volume but simple events, and they can even be somewhat stateful.

But fancier functionality probably needs to occur either in an existing μ-service or in a new μ-service you would design and implement.