6.S062 March 8 2017sp TinyDB

Prehistory -

Sensor networks - Berkeley, MIT, UCLA,

Unusual because of the degree of interest, fact that it produced real hardware that was widely available

Hardware was very constrained, which led to some idiosyncracies - very low rate radios, super highly constrained memory

(initial radio implementation had per-bit interrupts into software and 512 bytes of RAM!!!)

TinyOS / nesC -- TinyOS was abstractions for hardware
nesC was a programming language for building TinyOS apps

Ok, so what problem is TinyDB solving? Making it easy to program a large collection of devices as if they are a single system, without worrying about details of distributed coordination, as long as your app fits into the framework of distributed aggregation, which as we will discuss, is surprisingly general.

Steps:

- query dissemination
- data collection via routing tree
    - "partial aggregation" as data flows up

(walk through slides through demo)

- results displayed at root

<u>Query dissemination</u>

Flooding

Complex because we need to be somewhat controlled about it -- just sending over and over will overwhelm other nodes, and queries are multiple packets. E.g.,:
- set a random timer
- if don't hear the query from K other nodes before timer fires, forward packet
- combine with TTL (maximum number of retransmits) that is decremented after each forward

TinyDB combines flooding with "snooping" on packets -- if it hears about a query it doesn't know about, it will start running it

Tree formation protocol:

Packets travel from children to root

Formation happens via periodic route advertisements, as well as "snooping"

Each packet includes a nodeid, and a parentid, and an ETX to the root

Each node maintains a neighbor table :

nodeid ETX depth lastPktId parent

Nodes pick a node as their parent if the ETX of an overheard packet + ETX to the transmitting node < current ETX to parent.

This can lead to cycles due to stale ETX estimates -- e..g, node 1 advertises ETX of 1, node 2 hears it, picks node 1 as its parent, and advertises ETX of 2. Node 1 doesn't get packets through for a few attempts, updates its ETX to 3, and then decides to switch to node 2.

Node 1 shouldn't pick node 2 as its parent; since it has been forwarding node 1's packets it can avoid doing this.

Cycles of length > 1 can still arise; if a node is asked to forward its own data, it needs to re-select a parent.

Query evaluation

- Divide time into "epochs"

- During an epoch, each node sends an aggregation of it children's data and its data to its parent,

- Each message contains a "partial state record" -- e.g., a running count

- What is the advantage of this? If a node has N children, instead of sending N messages, it can send 1

- When can a node forward data?

(Whenever it wants, but that would waste the power savings)
- Need to wait for children! How to do that?

- divide "epoch" into slots
- synchronize time (how to do this?) --
    listen to parent's advertised offset til start of epoch
    set timer til start of epoch to sync with parents, assuming xmit is instantaneous

Unfortunately, MAC layer can delay packets, so extra tricky -- need a way to inject the time just before the packet is sent

- nodes listen during slot MAX_LEVEL-level and transmit during MAX_LEVEL-level + 1

Power saving:

- nodes can shut off their radios in any slot they don't need to transmit or send

What are semantics of aggregation functions? What functions can and cannot be expressed?

(show slides)

<u>What about GROUP BYs?</u>

Conceptually the same -- just maintain 1 PSR per group

Complicated if there are many groups -- but note that its OK to evict a "partial" group at any point, just will waste messages -- root can combine values together

<u>Lossy data & managing:</u>

- TinyOS uses link-level ACKs for reliability, but it is still possible to lose data
- End-to-end acks wouldn't really make sense in TinyDB (why not?)
    - Streaming semantics -- timeliness more important than completeness
    (other systems investigate reliable historical data collection)

- A bunch of tricks we can employ to improve reliability

(show slides)

TinyDB -- contrast with LEACH:

Very concrete set of computations -- as opposed to networking view where not willing to make any assumptions about nature of computationmputaiton

Constrained by practical reality of radio / hardware -- couldn't really control the transmit power, very focused on what we could with the hardware instead of an abstract model

of what *might* be possible

If doing a simple aggregate computation (e.g., an average over all data), the amount of work done by every node is the same, requiring no rotation of cluster heads!  We weren't really worried about the clusterhead problem when building TinyDB.

Why was TinyDB so influential?

- useful, real system
- lots of possible ways to extend the framework to compute certain types of aggregations more efficiently or reliably

(additional slides)