Resilient Sensor Network Query Processing Using Logical Overlays

Alan B. Stokes, Alvaro A. A. Fernandes, Norman W. Paton

School of Computer Science
University of Manchester
Manchester, UK

{a.stokes|alvaro|norm}@cs.man.ac.uk

MobiDE 2012
What We Have Studied

- Sensor network query processors (SNQPs) are brittle because nodes are resource constrained and therefore can fail.
- By adapting to failure we are likely to significantly increase the lifetime of a wireless sensor network (WSN).
- Increasing the lifetime is likely to increase the amount and accuracy of the data delivered.
- We use redundancy to mitigate the effect of node failure.
- By controlling levels of redundancy at compile time, we incur small runtime costs.
Basics: Sensor Networks

WSN

- Static heterogeneous motes
- Radio links
- Data acquired in the network
- Delivered to the gateway

Motes

- Multiple hardware platforms (e.g., micaz, telosB)
- Radio transmitter/receiver
- LEDs
- (Optionally) sensor(s)

Limited Hardware Capabilities

- Finite energy stocks (e.g., batteries).
- Low processing power
- Small memory (e.g., 128kb).
- Short radio range (e.g., 20m - 100m).
Basics: Multihop WSNs

(from http://blogs.salleurl.edu/networking-and-internet-technologies/tag/wnsn/)
SNQPs

- Deployments can be in very hostile or hard to reach locations.
- We want the most tuples returned for the deployment cost.
- In-network processing can exploit the less energy intensive computation cost in relation to radio transmissions giving an extended lifetime of the network.
- SNQPs generate query evaluation plans (QEPs) that include in-network operators.
- There are a few SNQPs in the literature (e.g., TinyDB [3], AnduIN [2], and SNEE [1]).
Inputs to SNEE

SELECT RSTREAM m.light, f.light
FROM meadow[NOW] m,
forest[NOW] f
WHERE m.light < f.light

% QoS expectations
% acquisition rate: 10 seconds
% delivery time: 600 seconds

Figure 1: SNEEql Query

meadow (id, time, temp, light)
forest (id, time, temp, light)

meadow sources: {6, 9}
forest sources: {7}
sink: {8}

Figure 2: Schemas

Figure 3: Connectivity Graph
SNEE Features/Assumptions

- SNEE generates lean QEPs in terms of energy expenditure and memory footprint.
- SNEE QEPs are brittle due to the assumption that the connectivity graph (CG) is stable.
- But the motes used QEP tend to fail, and when they do, the QEP may fail too.
How Does SNEE Generate QEPs?

**SNEE Compilation Stack**

- SNEE adopts the two-phase approach to the generation of distributed QEPs.
- The single-site phase executes classical query optimisation techniques.
- The multi-site phase handles specific WSN requirements.
- A final stage generates source code for in-network execution.

![Figure 4 : SNEE Compilation Stack]
SELECT RSTREAM m.light, f.light
    FROM meadow[NOW] m,
        forest[NOW] f
    WHERE m.light < f.light

% QoS expectations
% acquisition rate: 10 seconds
% delivery time: 600 seconds

Figure 1: SNEEql Query

meadow (id, time, temp, light)
forest (id, time, temp, light)

meadow sources: {6, 9}
forest sources: {7}
sink: {8}

Figure 2: Schemas

Figure 3: Connectivity Graph
Figure 4: Distributed Algebraic Form

Figure 5: Routing Tree (RT)

<table>
<thead>
<tr>
<th>Time (ms)</th>
<th>Sites</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>F₃₁</td>
</tr>
<tr>
<td>100</td>
<td>F₄₁</td>
</tr>
<tr>
<td>5000</td>
<td>F₃₁</td>
</tr>
<tr>
<td>5100</td>
<td>F₄₂</td>
</tr>
<tr>
<td>5401</td>
<td>tx₁</td>
</tr>
<tr>
<td>5702</td>
<td>tx₃</td>
</tr>
<tr>
<td>6002</td>
<td>tx₁</td>
</tr>
<tr>
<td>6302</td>
<td>F₂₁</td>
</tr>
<tr>
<td>6316</td>
<td>tx₈</td>
</tr>
<tr>
<td>6675</td>
<td>rx₁</td>
</tr>
<tr>
<td>7460</td>
<td>F₁₁</td>
</tr>
</tbody>
</table>

Figure 6: Agenda
Node Failure

**Goals**

- Rerouting of tuples around the failed node
- Relocation of operators on the failed node
- Adjustments to the agenda to reflect rerouting/relocation

**Consequences of adapting**

- Loss of time whilst adapting.
- Expenditure of energy used in adapting.
- Risk that an adaptation now may prevent a better one later on
Hypothesis, Approach and Aims

**Hypothesis**  The energy cost incurred in adapting to node failure is significantly outweighed by the benefits from the additional energy available as a result of adapting in terms of extended lifetime and therefore increased number of tuples delivered.

**Approach**  To use an overlay network where each logical node maps to $k$ equivalent physical nodes when redundancy permits.

**Aims**  
1. To avoid runtime reprogramming by precomputing equivalence classes
2. To simplify runtime decisions to adapt
3. To carry out adaptations cost-effectively
An overlay network abstracts aspects of the underlying physical network.

In our case, the abstraction is over a set of physical nodes.

We call the nodes in the overlay network *logical nodes* because one node in the overlay maps to several physical nodes in the WSN.

We call it a *(logical) overlay (network).*
Logical Overlay Strategy: Overview (1)

Compile Time

▶ Node redundancy is used to generate resilient deployments.
▶ Equivalence classes for nodes in the RT are derived.
▶ A logical node abstraction over the physical nodes is created.
▶ Physical nodes within a logical node contain the same QEP fragments (i.e., binaries).
▶ Only one node in a logical node is active at any point in time.

Run Time

▶ A node fails.
▶ Another physical node in the same logical node is selected and activated to replace the failed node.
▶ Tuples are redirected from the children to the new node.
Overlay Strategy: Compile Time

Compile-Time Steps

1. Generates a super-overlay network over the RT nodes
2. Generates a set of logical overlays from the super-overlay network
3. Determines which logical overlay give the largest estimated lifetime
4. Copies the code from each RT node to all other nodes in the same logical node

Figure 7: Logical Overlay Steps
Super-Overlay Generation (1)

Each active node $R$ in the RT is compared with every node $W$ in the WSN, excluding the other RT nodes.

If $W$ satisfies the following criteria w.r.t. $R$, then $W$ becomes a node in the same super-logical node as $R$:

1. $\text{freeMemory}(W) \geq \text{freeMemory}(R)$.
2. There is a connectivity edge between $W$ and $\text{parent}(R)$.
3. There is a connectivity edge between $W$ and every $\text{children}(R)$.
4. If $R$ is an acquisition node, then the deployer has asserted that $\text{equipotent}(R, W)$. 
Super-Overlay Generation (2)

Figure 8: CG and RT

Figure 9: Super-Overlay Network
Compile-Time Steps (cont.)

2. Generates overlay networks with different permutations of physical nodes for each node in the RT

Permutations have size at least $k$ so as to achieve a $k$-level of resilience, i.e., $k$ nodes must fail for the logical node to fail.

3. Estimates the cost/benefit of running the QEP over each overlay over a sequence of node failures

Figure 7: Logical Overlay Steps (again)
Figure 9: Super-Overlay Network (again)

Figure 10: Overlay Network
Compile-Time Steps (concl.)

4. The QEP fragment/agendas located on the RT nodes are copied onto all the other physical nodes within the same logical node.

Figure 7: Logical Overlay Steps (again)
Replication Within Logical Nodes (2)

Figure 10: Overlay Network (again)

Figure 11: Complete Overlay Network
Overlay Strategy: Run Time

Run-Time Steps

1. A physical node fails.
2. Another physical node from the same logical node is selected and activated that has the least total energy cost of communicating with the parent and children.
3. The tuples from the children of the failed node are redirected to the new active node.

This incurs an energy cost of transmitting one message down the RT.
Run-Time Selection and Redirection

**Figure 11**: Complete Overlay Network (again)

**Figure 12**: Overlay Network after Node 1 Fails
Experimental Evaluation

Goals

▶ To gain insights on the effect of a sequence of node failures
▶ To measure how many agenda cycles and how many more tuples delivered are made possible by adaptations
▶ To study how past decisions affect the lifetime of the network
Experimental Procedure

- 30 different topologies for each classical query type
- Each topology is run with its corresponding query
- Nodes fail uniformly throughout the original QEP lifetime.
- When a node fails, the system adapts
- A sequence of up to eight node failures
Additional Agenda Cycles

- Estimated lifetime increases between 9% and 83%.

Figure 13: Additional Agenda Cycles over a Sequence of Node Failures (SELECT * query)
Estimated lifetime increases between 9% and 83%.

Figure 14: Additional Agenda Cycles over a Sequence of Node Failures (SELECT * query)
Estimated increase in tuples delivered between 27% and 300%.

Figure 15: Additional Tuples Delivered over a Sequence of Node Failures (SELECT * query)
Related Work

- **Other SNQPs**
  - TinyDB can adapt to some node failures with its semantic routing tree but does not ensure 100% tuple delivery due to interference and tuple collisions.
  - AnduIN uses TCP/IP to handle transmissions between the acquisition nodes and the gateway which gives it some robustness to node failure but this inflates the binaries and limits how much in-network processing can be done.

- **WSN adaptive techniques.**
  - Multi-route communication unrealistically tends to assume cheap communication costs.
  - Cluster-based algorithms are unlikely to be as cost-effective because of the cost of run-time cluster-head selection.
Conclusions and Future Work

- Adapting to node failure can significantly increase the lifetime of a deployment, leading to more tuples delivered, and therefore, for aggregation queries, more accurate results.

- The loss of energy due to adaptations is small in comparison to the increase in lifetime from the additional energy made available by adapting.

- We are working on computing at compile time a successor relation over QEPs that expresses when to move from one QEP to another, thereby using load redistribution to prevent node failure.
Acknowledgements

Thanks to Ixent Galpin, Christian Y. A. Brenninkmeijer, Alasdair J. G. Gray, and Farhana Jabeen for their work on SNEE and for all their help. Thanks to EPSRC for funding Alan B. Stokes doctoral work.

References


How Does SNEE Generate QEPs?

Algorithm Selection

- At the end of Stage 3 we have the physical algebraic form (PAF) of the query.
- Each operator has a defined algorithm and execution order.

Figure 16: Physical Algebraic Form
How Does SNEE Generate QEPs?

Routing

- Stage 4 determines how to deliver tuples from the acquisition nodes to the sink node.
- An approximate Steiner tree is computed (it is a minimum spanning tree including a given set of nodes, i.e., the sink and the acquisition nodes).

Figure 17: Routing Tree
How Does SNEE Generate QEPs?

Where Scheduler

- Stage 5 determines which nodes are to execute QEP fragments.
- This is determined by heuristics designed to reduce tuple transmission.
- It results in the distributed algebraic form (DAF) of the query.

Figure 18: Distributed Algebraic Form
How Does SNEE Generate QEPs?

When Scheduler

- Stage 6 determines when nodes are to execute the fragments assigned to them in the DAF and when to transmit tuples between nodes whilst avoiding collision.
- It gives rise to a sequential ordering of fragment execution and data transmissions.
- The agenda is executed cyclically.

<table>
<thead>
<tr>
<th>Time (ms)</th>
<th>Sites</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>F3₁</td>
</tr>
<tr>
<td>100</td>
<td>F4₁</td>
</tr>
<tr>
<td>5000</td>
<td>F3₁</td>
</tr>
<tr>
<td>5100</td>
<td>tx₁</td>
</tr>
<tr>
<td>5401</td>
<td>tx₃</td>
</tr>
<tr>
<td>5702</td>
<td>tx₃</td>
</tr>
<tr>
<td>6002</td>
<td>tx₁</td>
</tr>
<tr>
<td>6302</td>
<td>F₂₁</td>
</tr>
<tr>
<td>6316</td>
<td>tx₈</td>
</tr>
<tr>
<td>6675</td>
<td>F₁₁</td>
</tr>
<tr>
<td>7460</td>
<td>sleeping 2540)</td>
</tr>
</tbody>
</table>

Figure 19: Agenda