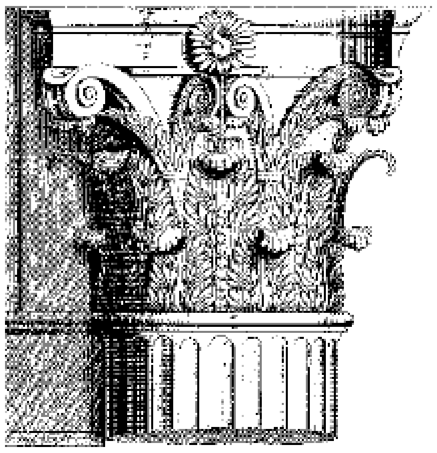

The Appia topology solver

Implementation

Li-Shiuan Peh

This document describes the solver algorithms implemented for Appia and the evaluation results.

Introduction

This document describes the Greedy and Merge heuristics implemented in Appia and the experiments done to evaluate their coverage and optimality.

Table of contents

The Greedy heuristic

The greedy heuristic starts with an empty fabric and then tries to add a flow at a time, in the cheapest way possible given its knowledge at the node. Hence, it first tries to add a point-to-point link, since that only incurs the cost of a fibre. If this is not possible due to degree constraints, it tries to use a hub, and if that is again not possible, it resorts to using a switch. The heuristic fails if none of these alternatives are possible, given capacity and degree constraints.

Algorithm:

```
Add a point-to-point link
If degree constraints is violated
  Share another link's internal node
  If not,
    Share a fibre with another link, and add a new internal node
  If not,
    Upgrade an existing hub to a switch.
  If not,
    Try to coalesce existing links so as to free up a port for this new link
  If not,
    Fail.
```

The Merge heuristic

Instead of starting with an empty fabric and incrementally building it, the Merge heuristic starts off with a complete fabric, with each flow represented as a point-to-point link between nodes.

Also, instead of having terminal nodes which represents hosts and devices, the Merge heuristic creates a terminal node for each port of a host and device. Flows are then distributed among these ports.

Hence, when there is more than one flow to a port, the flows must be merged and coalesced into a single link (fibre).

Algorithm:

```
Create a node for each port of a host/device.
Distribute the flows of a host/device to the ports by
   Picking the largest flow and assigning it to the port with the minimum load.

For each port
   If number of flows at the port is greater than 1
       Merge flows into an existing internal node or a new internal node

Bind the cheapest possible switch or hub to the internal nodes.

For each internal node
   Merge other internal nodes into it if possible
```

Evaluation

In our evaluation, we used 9 configurations, each with different numbers of hosts, devices and maximum number of streams per host. 20 assignments are then randomly generated for each configuration, each of these assignments having randomly generated number of streams per host (subject to the maximum), randomly generated bandwidths for each stream, and randomly generated device destination for each stream. The configurations range from 3 hosts with 3 devices, up to 50 hosts with 50 devices.

The experiments assume hosts and devices with 2 ports each, and hubs and switches with 20 ports each.

Coverage

We collected statistics of the number of successes of the heuristics, i.e., how frequently do they manage to generate a feasible fabric.

Assignments may fail if they are infeasible to start with, say if the total stream bandwidths exceed ports' capacities at a host/device. This is pruned out in the generation of test cases. Assignments may also be feasible if there is no way to divide the streams among the ports. For instance, if we have two ports, and three streams of

90MB/s, 90MB/s and 20MB/s each, although the total streams' bandwidth = 200MB/s can be sustained by the 2 ports, there is no way to apportion these streams into the 2 ports such that each port's individual capacity of 100MB/s is not violated. This kind of infeasible assignments are NOT pruned in the assignment generation stage. It should be done, and can be done by doing an exhaustive allocation of the streams to ports, since the number of streams per port and the number of ports are usually small.

Of course, assignments may fail due to the algorithm's fault. The greedy heuristic for instance may fail because none of the local changes it proposed can be used without violating constraints. The Merge heuristic will fail if none of the fabric elements supplied can handle the capacity and degree needs of the internal node.

Out of the feasible assignments among the 180 test assignments, Greedy is able to find a solution in 46.6% of the time, and Merge is successful 93.3% of the time.

Optimality

Since we do not have an optimal benchmark, we had to resort to manual design as a comparison.

I hence hand-pick 2 assignments from the first 6 configurations, and manually design these 12 assignments. The cost of my hand-drawn fabric is then compared with that generated by the two Appia solvers (when they succeed) and it is found that out of 10 assignments which Greedy succeeds in, its fabrics costs $5,610 more than the hand-drawn fabrics on average. For the Merge heuristic, its fabrics cost $2,658 more than the hand-drawn fabrics on average, for all the 12 assignments.

We also compared the Merge heuristic and the Greedy heuristic, and found that out of the 69 assignments which Greedy succeeded in, its fabrics cost on average $2,785 more than that of Merge's.

Future Work

Random Greedy and Random Merge

We can add randomness to the Greedy and Merge heuristics. For Greedy, the order of traversing nodes and links can be randomized. For Merge, the distribution of flows to ports and the order of traversing ports can be randomized. The best result from a couple of cases can then be selected.

Port violation guiding Greedy heuristic

We can zero in on the node which is most congested, i.e., the node where the number of ports needed - the number of existing ports and apply the Greedy heuristic to that node first. This may help is improving the coverage of the Greedy heuristic.

Multi-layer hubs and switches

The Greedy and Merge heuristics can be extended to handle multi-layer cascaded hubs and switch fabrics.

Ports for hosts can be increased, subject to a bound

The number of ports on hosts and devices need not be a constraint, since if there are enough slots on hosts, additional adapters can be bought. Hence, Appia should be able to consider the costs of purchasing an adapter versus that of purchasing fabric elements in its decision.

Incremental solver

It should be possible to pass Appia an existing fabric, new additional flows, and to want a new fabric which incurs the least ADDITIONAL cost whilst satisfying the new flows.

Take existing elements into account

In practice, it may be that users already own a hub or a switch, and hence, that should be part of Appia's decision in choosing between fabric elements.

Last modified: Fri Sep 18 15:30:02 PDT 1998

Li-Shiuan Peh <lspeh@hpl.hp.com>