Run, Fatboy, Run:
Applying the Reduction to Uniprocessor Algorithm to Other Wide Resources

Andrew G. Shewmaker Carlos Maltzahn Scott Brandt
University of California Santa Cruz
Katia Obraczka Ivo Jimenez

Abstract

The RUN (Reduction to UNiprocessor)[3] algorithm was first described by Regnier, et al. as a novel solution to real-time multiprocessor scheduling, but its ideas can be applied to other scheduling problems involving arrays of similar resources. This technical report briefly describes how RUN can improve the management of network routes, queueing disciplines, disks, and batch schedulers.

1 Introduction

The RUN algorithm takes advantage of two features of highly loaded systems. First, a busy system will generally have many small tasks that can be packed together and treated as one task. Second, a highly loaded system has little idle time, so it makes more sense to solve the dual schedule (i.e. when tasks aren't running).

In a system with N processes and M processors where each process requires a fixed share of a processor, packing shrinks the size of N and taking the dual of the system reduces the size of M whenever $N < 2M$. By alternating packing and dual operations, Regnier, et al. showed that they were able to reduce the difficult multiprocessor problem down to a simple uniprocessor problem. The approach is revolutionary because it is simple and provably more efficient in terms of context switches and migrations than any previous approach.

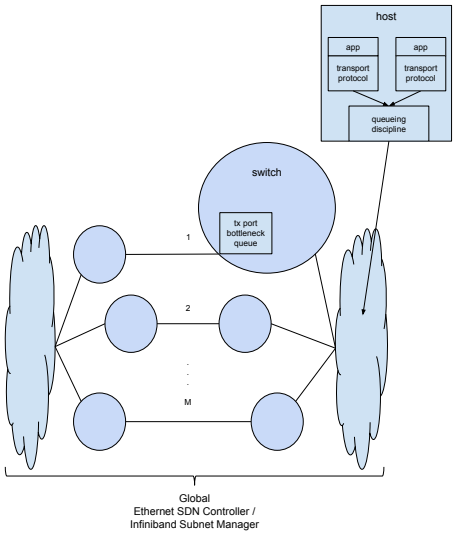
RUN was designed with RAD (Resource Allocation and Dispatching)[1] reservations in mind, but it can be incrementally deployed with common rate-limiting techniques such as TBFs (Token Bucket Filters), fair share, and weighted queueing. However, in order to benefit from the full power of RUN, subsystems will need to adopt full RAD reservations.

RAD reservations are (utilization, period) tuples that obsolete priority classes and previous rate-limit specifications. Current methods possess a limited number of relative, coarse-grained classes (priorities), require rates to be strictly satisfisfied for any measured interval (TBFs), have common periods between all tasks, or have a fixed linear mapping between periods to priorities. RAD reservations enable arbitrarily fine-grained QoS (Quality of Service), possess meanings that stay consistent in a dynamic environment, and allow straightforward reasoning about composing end-to-end QoS.

2 Networking

Providing QoS on networks is complex because several independent queues must be managed in concert: transmission and reception queues on the communicating hosts and the transmission queues on the bottleneck switches. A flow is affected by the route it takes, the bottlenecks on that route, and the manner in which a host sends its data.

Figure 1: Network QoS Layers



Note that even with a global scheduler ensuring routes aren't overloaded combined with rate limiting on hosts that bottleneck queues can still build up and drop packets. The bottleneck would have to be able to handle the worst case simultaneous burst from every flow on that route. In order to minimize bottleneck queue usage, the transport protocol still needs to adapt to congestion.

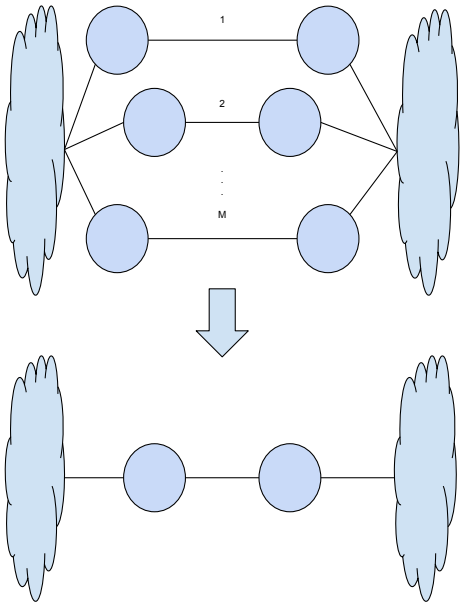
2.1 Routes

The edges of networks present a perfect opportunity for RUN. Whether it is a global WAN gateway where bandwidth is extremely limited and precious or the high performance interconnect between a supercomputer and a parallel filesystem, flows should maximize the utilization of the available routes while preventing congestion and data loss.

As opposed to traditional distributed multipath routing approaches described by Hopps [2], RUN would be used by a SDN (Software Defined Network) Controller or a Subnet Manager (in the case of Infiniband) to assign routes to flows after they have passed a simple admission control test: Does the new flow's rate cause the total flow rate to exceed capacity?

To be clear, RUN is not discovering the topology of the network. It is scheduling the routes given to it to manage. Also, it would take further work to make RUN take considerations other than a path's cost (e.g. security) into account.

Figure 2: Reduce to Unipath



2.2 Host Hardware Queues and Qdiscs

Multiple hardware queues present another opportunity for the RUN algorithm. It has provably low numbers of migrations, and can prevent head-of-line blocking while enforcing QoS. Current Linux support for multiqueue network hardware allows an administrator to pin cores or NUMA nodes to a queue. While that minimizes context switches and maximizes cache use, it suffers from head-of-line blocking. Alternatively, a round-robin scheduler can be used. While being fair and avoiding head-of-line blocking, round-robin necessarily hurts efficiency.

In addition to the multiqueue support already mentioned, a network classifier cgroup can tag packets to be handled by specific qdiscs. Combining existing strict-rate limiting qdiscs, routes scheduled by RUN, and bottleneck switches with enough buffering to handle worst case simultaneous bursts from each flow, the network can guarantee performance without packet drops.

Worst case buffer requirents change with different numbers of flows, so providing adequate memory in bottlenecks might not be feasible. The variability in delay in the worst case might also be undesired. Instead of worrying about providing enough memory for the worst case, we should make transport protocols adapt to increasing round trip times due to filling buffers.

3 Storage

In general, RUN cannot be applied to arrays of storage devices since content is not replicated everywhere. However, RUN could be useful in two scenarios.

3.1 Placement

When the question arises where to store new data, RUN can help manage performance after other considerations, such as available space are answered.

3.2 Replicas

Parallel file systems often replicate data between multiple servers. Usually one server is considered the primary, but it can become overloaded and want to balance its client load with its replicas. As long as consistency among replicas is maintained (trivially true for read-only access), then RUN can be used to schedule use of the replicas.

Conclusion

This tech report only briefly explores how the Reduction to Uniprocessor algorithm can be applied to other resources. RUN is a simple algorithm with immense

power, and it should enable comprehensive QoS across many layers of resources. In particular, the Radon Network QoS project will be implementing and evaluating RUN in the near future. Please stay tuned.

References

[1] Scott A. Brandt, Scott Banachowski, Caixue Lin, and Timothy Bisson. Dynamic integrated scheduling of hard real-time, soft real-time and non-real-time processes. In *Proceedings of the 24th IEEE Real-Time Systems Symposium (RTSS 2003)*, pages 396–407, December 2003.

[2] Christian E Hopps and Dave Thaler. Multipath issues in unicast and multicast next-hop selection. 2000.

[3] Paul Regnier, George Lima, Ernesto Massa, Greg Levin, and Scott Brandt. Run: Optimal multiprocessor real-time scheduling via reduction to uniprocessor. In *Real-Time Systems Symposium (RTSS), 2011 IEEE 32nd*, pages 104–115. IEEE, 2011.