Voltage-Aware Load Balancing For Series-Connected Server Clusters

Josiah McClurg

***Abstract*—Making use of series-connected voltage domains in today's data center is an exciting area of research for more than the mere potential of orders of magnitude increases in data center efficiency. Just this past year, there have been several experimental proof of concept designs, demonstrating some level of practical feasibility of both software (load-side current control) and hardware (differential power processing) regulation strategies for series-connected digital loads. As a step towards improving the state of the art in voltage aware load balancing, this project presents a flexible software framework for distributed load balancing.**

I. Introduction

There is little debate as to whether or not the series-connected power distribution architecture offers a large theoretical potential for efficiency improvement. Moreover, hardware regulation strategies have demonstrated some measure of practical success, both in the area of low-power [1] and more recently high-power [2] digital loads. However, hardware regulation does require additional infrastructure investment, while a load-side regulation strategy can be implemented in software and thus might be capable of maintaining server voltage "for free," as it were. This is a particularly compelling option, given the availability of sophisticated power-aware scheduling algorithms [3] and the existence of proof-of-concept software-only voltage regulation implementations [4].

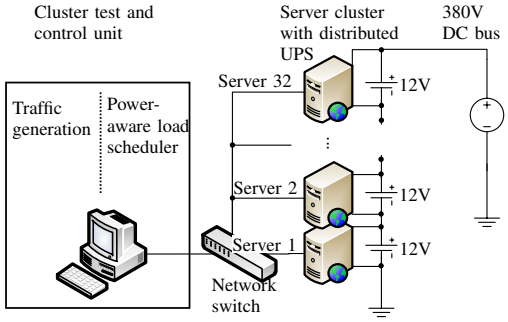
Existing weighted load-balancing protocols [5] may be capable of achieving similar functionality through careful coordination and continuous adjustment of traffic allocation weights. However, this approach is costly in terms of messages and may thwart some system-level goals for load allocation. The protocol presented here was designed to operate at a small scale (the level of an individual server rack), and to avoid deviating from the pre-imposed load allocation except when necessary to prevent an overvoltage.

The primary motivation this software framework and the voltage-aware load balancer presented is to serve as the next step beyond the centralized load balancing software presented in [4]. A graphical illustration of the desired architectural changes is shown in Figure I.

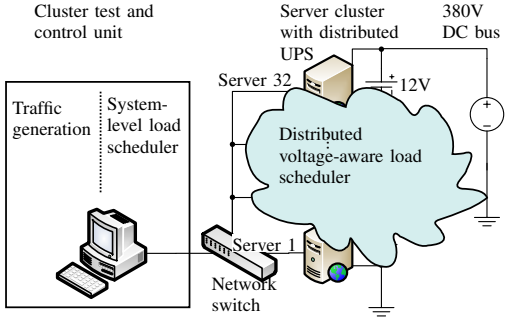
II. Background

Assuming that each server has some input capacitance C_i, then its voltage depends upon the string current I_s and the server current I_i according to

$$V_i(t) = \frac{1}{C_i}\int_0^t \left(I_s(t) - I_i(t)\right) dt + V_{i0}$$



(a) Existing system architecture



(b) Improved system architecture

Fig. 1. Motivation for distributed voltage-aware load balancer.

Further assuming that the average string current and the average server current are constant (we will justify this assumption later) so that $\frac{1}{t}\int_0^t I_s(t)dt = \langle I_s \rangle$, and $\frac{1}{t}\int_0^t I_i(t)dt = \langle I_i \rangle$ then the relationship becomes a simple linear one:

$$V_i(t) = \frac{\langle I_s \rangle - \langle I_i \rangle}{C_i} t + V_{i0}$$

Now, the electrical constraint that $\sum_{i=0}^{N} V_i(t) = V_{bus}$ may actually be ignored in the case where either I_s or N is large. In the first case, nonzero line resistance comes into play, and allows mismatch between the sum of the server voltages and the bus voltage. In the second case, $\sum_{i=0}^{N} V_i(t) \pm \max(V_i(t)) \approx V_{bus}$, so the difference between the voltage calculated with the KVL constraint and the voltage calculated only using KCL is small. Because both our system exhibits both high currents and a large number of nodes, it is proper to continue with the derivation, using KCL only.

Because of the linear nature of our system, a simple proportional control law will suffice to reach nominal voltage V_{nom} in time Δt:

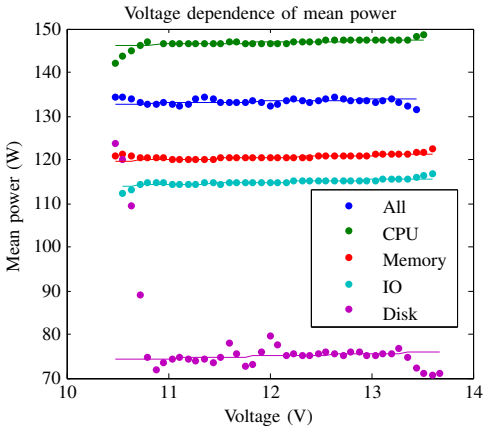


Fig. 2. Desktop server power under various benchmarks

$$\langle I_i \rangle_{setpoint} = \langle I_s \rangle_{measured} - (V_{nom} - V_{i,measured}) \frac{C_i}{\Delta t}$$

Electrically, a server can be modeled as a constant power load, with the current draw depending almost solely on load and frequency. This behavior is clearly illustrated by Figure 2, which shows the measured power output of a small desktop computer, running at constant frequency, under various input voltages and workloads.

The physical explanation for this behavior is that the majority of the load current is being drawn from constant-voltage CMOS logic. According to the well-known CMOS power consumption formula $P \approx \alpha C_L f V^2$, this equates to a load current which depends only on f and the logic utilization factor α. Because this load is being supplied by a high-efficiency DC/DC converter, the input power remains relatively constant over the operating voltage range. In practice, the activity factor α is mostly determined by CPU load. To illustrate this point, consider the benchmarks of Figure 2. Compared to the power consumption of the green CPU-only benchmark, the mean power is actually reduced when the software attempts to load the peripherals in addition to the CPU. Even with all the other peripherals loaded, the slight reduction in CPU utilization induced by the peripheral latency overhead was enough to significantly reduce total power consumption.

Now, the end goal of this protocol will be to obtain a local estimate of the global state variable $\langle I_s \rangle$, and to set $\langle I_i \rangle$ to a desired setpoint $\langle I_i \rangle_{setpoint}$ by rerouting queued server traffic within the cluster. However, because we have shown that current is directly proportional to CPU utilization, it is sufficient (for control purposes) to provide a means to throttle or increase the average CPU time.

The way that this implementation accomplishes this is by adjusting the number of concurrently executing threads ("jobs" in the sequel). Systems which are memory or disk constrained (i.e. most real-world systems) will show an increased CPU utilization as the thread count increases. Beyond a certain point, this may actually come at the expense of an overall increase in latency, and may even result in thrashing if the thread number is allowed to grow without bound. But, these considerations are beyond the scope of this implementation and are often avoidable by imposing a simple maximum thread count constraint.

III. Design goals

In the process of our design, we make the following assumptions:

1) Large C_i: Inter-server communication can happen during input voltage transients. This is certainly true in the presence of a distributed uninterruptible power supply (our current configuration). More research is needed to determine if this is a good assumption in the general case.
2) Bursty traffic: Requests are sent to the cluster in large bursts, rather being spread out as individual requests. This assumption is needed only to ensure that there are enough jobs available for everyone that needs them.
3) Fast and reliable communication channel between servers. This is a good assumption for many server cluster configurations (local Ethernet with dedicated switch).
4) As a corollary to the above, job transfer between nodes is much faster than the execution of any job.
5) External group membership management detects node failures/group leaves and group joins, and notifies group members. This is done for implementation convenience only, as the protocol is easily extensible to a distributed failure detect.
6) Time-invariant relationship between number of parallel jobs and CPU current. Future implementations may easily relax this assumption, because the protocol supports an arbitrary job allocation set point from the voltage control loop.

Under these assumptions, we were able to meet the following design goals:

1) Minimal impact on throughput and latency: Protocol only re-allocates jobs which otherwise would have had to wait to complete.
2) Minimal re-allocation of jobs: Protocol only re-allocates jobs if one or more servers is in danger of overvoltage.
3) No job duplication, loss of jobs, or failure to acquire needed and available jobs: Protocol implements a distributed mutex (Ricart Agrawala) to manage access to job allocation routines.
4) Completely thread safe operation: Each server supports multiple simultaneous job consumers (simulating many-core processor) and multiple simultaneous job producers (simulating multiple parallel load balancers).

IV. Protocol

Algorithm 1 Job allocation: (called when the queue size decreases, or the batch size [number of required parallel jobs] changes and queue size $<$ batch size)

```
1: procedure POPULATEQUEUE
2:     Acquire local locks (thread safety)
3:     Acquire distributed mutex (implemented as Ricart Agrawala)
4:     for Each other node do
5:         Get number of extra jobs (queue size - batch size)
6:     end for
7:     Based on this information, decide where to get jobs.
8:     Get them according to the previous step.
9:     Enqueue the new jobs, marking them with a higher priority (to prevent multi-hopping of jobs).
10:    Exit distributed mutex
11:    Exit local locks
12: end procedure
```

V. Compiling and Running

1) install apache thrift with libevent support
2) git https://github.com/jcmcclurg/jcm-loadbalancer.git
3) make
4) To start the multicast group manager, type: ./hub
5) To start individual servers, type: ./node
6) To test the operation of the protocol, type: ./nodeTest

The node test executable demonstrates the correct operation of the protocol by sending many jobs to one node only, and directing the voltage control loop to indicate that all nodes need to execute three jobs in parallel to avoid an overvoltage condition. The jobs are slowed down significantly, so the operation of the protocol can be observed by the user. What will be seen is that one node will enqueue many jobs, while the other nodes continually re-allocate those jobs to themselves, three at a time, as is needed to maintain correct CPU loading. To exit the nodes, wait until you see "Job consumer processing...", and then hit Ctrl+C. To exit the hub or nodeTest, simply hit Ctrl+C.

VI. Conclusion and Ongoing Work

As the next step toward a fully operational series-connected server cluster, this project has designed and implemented a fully distributed and flexible load balancing protocol which lends itself particularly well to voltage regulation. The protocol is designed to operate with minimal intrusion into the existing traffic patterns – only performing job reallocation when it becomes necessary to do so to prevent an overvoltage condition. Future research will see the performance of this implementation first tested on server hardware instrumented with power monitoring equipment. Following application-specific parameter tuning, the protocol will be installed and tested on a series-connected server cluster, comparing performance with the existing centralized scheduler.

References

[1] P. S. Shenoy and P. T. Krein, "Differential power processing for DC systems," *IEEE Transactions on Power Electronics*, vol. 28, no. 4, pp. 1795 –1806, Apr. 2013.

[2] K. Kesarwani, C. Schaef, C. R. Sullivan, and J. T. Stauth, "A multi-level ladder converter supporting verticallystacked digital voltage domains," in *Applied Power Electronics Conference and Exposition (APEC), 2013 Twenty-Eighth Annual IEEE*, 2013.

[3] V. Sharma, A. Thomas, T. Abdelzaher, K. Skadron, and Z. Lu, "Power-aware QoS management in web servers," in *Real-Time Systems Symposium, 2003. RTSS 2003. 24th IEEE*, 2003, p. 6372.

[4] J. McClurg, Y. Zhang, J. Wheeler, and R. Pilawa-Podgurski, "Re-thinking data center power delivery: Regulating series-connected voltage domains in software," in *Power and Energy Conference at Illinois (PECI), 2013 IEEE*, 2013, pp. 147–154.

[5] M. Randles, E. Odat, D. Lamb, O. Abu-Rahmeh, and A. Taleb-Bendiab, "A comparative experiment in distributed load balancing," in *Developments in eSystems Engineering (DESE), 2009 Second International Conference on*, 2009, pp. 258–265.