Go, Server, Go! Parallel Computing with Moving Servers

R. Barber G. Lohman R. Mueller I. Pandis V. Raman W. Wilcke

IBM Almaden Research Center, San Jose, CA

Abstract

In data centers today, servers are stationary and data flows on a hierarchical network of switches and routers. But such static server arrangements require very scalable networks, and many applications are bottlenecked by network bandwidth. In addition, server density is kept low to enable maintenance and upgrades, as well as to increase air flow. In this paper, we propose a design in which servers move physically, and communicate via point-to-point connections (instead of switches). We argue that this allows data transfer bandwidth to scale linearly with the number of servers, and that moving servers is not as expensive as it sounds, at least in terms of power consumption. Moreover, while servers move around, they regularly reach the perimeters of the system, which helps with heat dissipation and with servicing of failed nodes. This design also helps in traditional switch-based networks, to improve density and maintainability.

1 Introduction

"Sneaker networks" have long been used for wide area data transfer. They give better throughput, and in many cases better reliability than WANs. As a classic example, Jim Gray used to ship tapes via FedEx [1]. This paper explores networks of moving computers on a LAN as a way to avoid the switch bottleneck in parallel computing. In addition, moving servers can allow increased server density, because during movement servers reach the physical perimeter of the system. This aids uniform cooling and allows easy replacement of servers.

SOCC'13, October 01–03 2013, Santa Clara, CA, USA.
 ACM 978-1-4503-2428-1/13/10...$15.00.
http://dx.doi.org/10.1145/2523616.2523634

Communication Bottleneck for Data Shuffling. Communication is the limiter in most parallel systems. It shows up in many ways: network bandwidth, memory bandwidth, NUMA, latency issues, etc.

Our focus is on bandwidth alone, and with our solution – physical node movement – point-to-point latency is very high, on the order of *seconds*. This is okay, because we are targeting a specialized, yet common communication pattern – *data shuffling*. We have a set of nodes that want to shuffle data among themselves, and are willing to do so in a carefully coordinated fashion. Data shuffling is common in databases (joins, group-by, building indexes, clustering data prior to load, etc) and in map-reduce (to send data from mapper nodes to reducer nodes). In these problems, latency of a few seconds is okay, because bandwidth is the limiter.

This weakness of bandwidth versus computation is not accidental. Compute power increases with the transistor density, while network bandwidth is limited by the "perimeter" of the processing elements – pins, wires, NICs, and by switch bandwidths. Overall, the speed of parallel programs is limited by three technologies:

- *Processing capability*: the number of cores, clock speeds, the SIMD register width, etc.
- *Local bandwidth*, i.e., the speed at which data can enter or leave a processing element.
- *Aggregate bandwidth* at which the nodes can communicate, i.e., the bisection bandwidth of the network.

Parallel systems improve over time according to the slowest of these three technology trends. Today, network bisection bandwidths are typically the limiter.

As an example, consider what it will take to sort an exabyte of data using current technology[1]. Currently the largest published sort is a 10PB sort by the Google infrastructure team[2]. They presumably have access to very sophisticated networks and switches. Yet, the sort took 6.5 hours, and this was 11.7x the time taken for a 1PB sort. At this rate, sorting an exabyte will take a month.

[1] We use sort as an example of a parallel computing problem with a significant data shuffling step.

[2] See http://googleresearch.blogspot.com/2011/09/sorting-petabytes-with-mapreduce-next.html. Note that this reference is from Sept. 2011, so Google probably has improved results.

Of course this is not the full story. Supercomputers have long had specialized high-bandwidth interconnects, and optical interconnects promise very high bandwidths (e.g., [8]). The networking community is also investigating how to build high-bandwidth networks from commodity parts (e.g., see [3, 5, 7]).

Moving Servers. This paper presents an alternative scheme, based on moving the servers themselves. Of the three factors limiting parallelism, server movement eliminates the switch (last bullet above). Local bandwidth is still a factor, and node-to-node coupling becomes more challenging when nodes are moving. But we argue that current technologies such as capacitive coupling are sufficient; the nice thing is that, unlike a switch, it scales naturally with the number of nodes.

The reader may ask: why bother? After all, moving computers sounds complicated, expensive, and error prone. Why move computer nodes instead of moving electrons or photons (on the cable)?

First, there is the usual sneaker network argument:

$$Bandwidth = density \times movement\ frequency$$

For example, a node with 64 GB of storage that can travel across the cluster in 10 seconds has an effective bandwidth of 6.4 GB/s. And the aggregate bandwidth increases linearly when more nodes are added, or when storage density improves – we need not wait for the switches to improve. Photonics will soon provide very high bandwidth links, even over long distances. But they do come at high cost and complexity, so using high-bandwidth copper over short distances, and a sneaker network over long, may be simpler and more economical.

Second, we argue that moving servers is not as hard as it sounds. In Section 3 and Section 5 we present two high-level designs for moving nodes for which the power consumption for movement is less than 1% of the power needed for the electronics on the nodes. Also, when nodes are moving, all nodes regularly reach the periphery of the system. This has advantages for densely-packed configurations, because it allows for easy replacement of failed nodes. In fact, we think even traditional switched networks can be built this way (out of moving servers), and propose a design for this in Section 6.

Caveat: The authors are not mechanical engineers, and we have little experience building things at this scale, with so many moving parts. So please take our design with a spoon of salt. We have tried to list both the promise and pitfalls of this approach, but have surely missed some.

2 Overview

Servers that physically move present many challenges, most of which cannot be addressed in much detail within the limited scope of this paper. Traditionally, servers are powered and connected with wired networks, but the connectors could wear out if repeatedly pulled out and plugged back in. In a wireless world, then, how would servers get power and communicate with each other, and how would they be moved with respect to each other? We give a brief overview of each in this subsection, but focus mostly on the last topic in the remainder of this paper.

The operation we focus on is *data shuffling* (e.g., [6]): each node has some data to send to every other node. All nodes do the shuffle in lock-step, and we assume that this synchronization is done via conventional networks.

Getting power to moving nodes. Alternatives to power cables include power induction and the movement infrastructure itself. Capacitors can be used to deal with any power interruptions (e.g., when transitioning across movement rails or between induction field zones).

Communication between adjacent nodes. Wireless LANs have known limitations in bandwidth and overhead to establish connections. Instead, we propose to use capacitive coupling [4], in which each node has a ceramic-coated face with a metallic plate. The nodes communicate by coming near each other until the faces touch. Prototypes of capacitive coupling in the Intelligent Bricks project [9] at IBM have demonstrated bandwidth of 10 Gb/s bi-directional over a $4\,\text{cm} \times 4\,\text{cm}$ face.

Physically moving nodes. We present two schemes for moving nodes: (1) repeated start-stop motions interleaved by communication, and (2) continuous movement. In both, the data transfer is similar to that in hypercube networks – nodes containing data are arranged in a grid of columns and rows, and, in the 3D case, vertical levels. Shuffling then happens in multiple phases (hops):
P1: Each data item is shuffled to its destination column.
P2: Each data item is shuffled to its destination row.
P3: Each data item is shuffled to its destination level.

3 Shuffling by start-stop rotation

The scheme of this section involves repeated starts and stops of nodes. In each phase, pairs of adjacent rows (columns, levels) rotate, and every time two nodes in adjacent rows are positionally aligned, they stop and exchange data. Section 5 gives an alternative in which nodes move at fixed speed, except when turning around at corners.

3.1 Two-Row Shuffle by Half-Rotation

The primitive that we build on is a *Two-Row Shuffle*. It takes two adjacent rows of N nodes each, and shuffles each item to *either* of the nodes in its destination column.

We do this by moving the nodes clockwise, in a "half rotation". This takes $2N$ steps, as shown in Figure 1. Al-

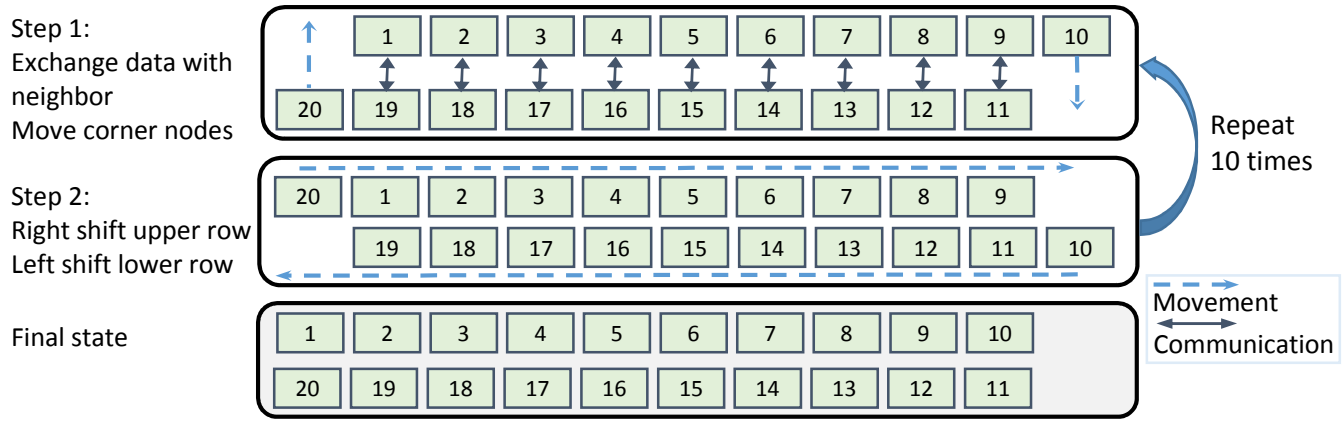

Figure 1: Two-Row Shuffle

ternate steps involve data exchange between neighbors, or node movement. During the data exchange step, the corner nodes also move vertically, to get the rotating effect. To see why this works, notice that every odd-numbered node will become adjacent to every other odd node during the rotation (similarly for the even nodes). Also observe that in the final state, every column has an odd node (and an even node). So all the data items of an odd node A that should end up in a particular column can be transferred to the corresponding odd node, when A is adjacent to that node. Data from the even nodes is shuffled similarly. Data is never transferred between even and odd nodes, because we only want to get data items to the correct column – we do not care which row.

Mechanical details. A key challenge is how to physically move nodes. Will this movement keep up with the data transfer speeds? Will it consume too much power? Will it introduce unreliability in the electronics?

We propose a simple scheme. The half-rotation above involves N movement steps, interleaved with N communication steps. In each movement step, all the nodes undergo constant acceleration for half the time, and corresponding deceleration for half the time. Then the nodes are at rest, and communicate with their neighbors. Since all the nodes in a row move the same way, they can all be attached to a singe drive (e.g., a rail or a belt), and the acceleration done via the drive (the corner node has to be detached, e.g., via an electromechanical solenoid).

Suppose each node is of size $s \times s$, with a gap of c between nodes. Then movement time per step, t, is:

$$\tfrac{1}{2}(s+c) = \tfrac{1}{2}a\left(\tfrac{t}{2}\right)^2$$

(where a stands for acceleration), because we accelerate for $t/2$ time and decelerate symmetrically for $t/2$ time.

The half-rotation involves N such steps, taking time $N \times t$. The data transfer in aggregate takes time d/b,

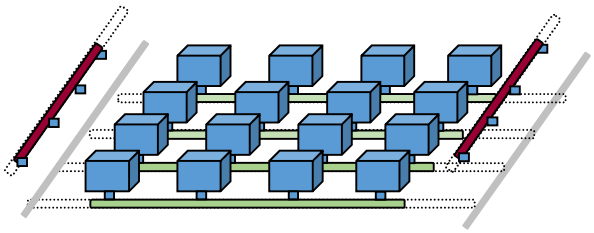

Figure 2: 2-D network (rails for phase 1).

where d is the data per node and b is the bi-directional face bandwidth between adjacent nodes.
So the overall time for the shuffle is:

$$\text{Shuffle time} = Nt + d/b$$

Also, the power consumption to move each node is:

$$\text{Power per node} = ma(s+c)/(2t) \text{ (on average)}$$

where m is the mass (as to friction, please see below). The factor 2 is because the deceleration is done by braking. Further improvement is possible via regenerative braking.

3.2 Extension to 2-D and 3-D

This scheme extends to two dimensions in a direct way. Arrange the nodes into an $N \times N$ grid. The shuffle occurs in two phases (hops). In Phase 1, pairs of adjacent rows do a half-rotation. This gets each data item to a node in its destination column. In Phase 2, pairs of adjacent columns do a half-rotation. This gets each data item to its destination row in its destination column.

Drivetrain. For each pair of adjacent rows, in Phase 1, we need a drive that repeatedly accelerates one row to the left and the other to the right. This could attach magnetically to the nodes, so that the corner nodes can be easily detached. Phase 2 requires another set of drives along the y axis (e.g., in phase 1 the drive can attach

below the nodes, and in phase 2 above the nodes). Note that in both phases, the movement of the corner nodes is synchronized across all pairs of adjacent rows (columns). So a single drive is sufficient to accelerate and decelerate all the corner nodes. Figure 2 shows this in more detail.

An alternative scheme is to use one drivetrain for all the odd rows, and one for all the even rows (and similarly for the columns), but this does not extend easily to 3-D.

Performance. The performance of this 2-D grid is similar to that of a 2-D hypercube. Since it involves two 1-D-like phases, the shuffle time is twice as much as 1-D:

$$\text{Shuffle time} = 2 \times (Nt + d/b)$$

Conceptually, the extension to 3-D is direct: we just add another phase. For example, the shuffle time becomes $3 \times (Nt + d/b)$, to shuffle data of size $N^3 d$, giving dramatically better density and shuffle speed. The drivetrain for 3-D is more complex, and is discussed later.

Table 1 lays out all these parameters, for a few configurations. We start with clusters of smartphone-like nodes, each of dimension 8cm×8cm×8cm. We consider two kinds: thin nodes that weigh 100 g, and thick ones that weigh 500 g (for reference, a Raspberry-Pi weighs 50 g).

As the table shows, we can sort 100PB in 420s (3D) and 1PB in 280s (2D), – equivalent to bidirectional switch bandwidths of 238 TBps and 3.5 TBps. To compare this to a traditional system we examined an actual cluster of 36 racks each hosting 20 2RU servers, each with dual ported 10Gbe links to a top-of-rack switch, each of which has 4 optical 40Gbe lines to a central cluster switch (2.5x oversubscribed). Both the cluster and top-of-rack switches tout full cross-bar bandwidth, so this 720-node cluster supports 1.44 TBps among servers within a rack, and 576 GBps across racks. Assuming a shuffle with uniform data transfers with no overlap of intra-rack and inter-rack transfer, we get an aggregate bandwidth of 585 GBps, which can do a 1 PB shuffle in 28 min. The network equipment costs around $2.6M ($750 per NIC, $35K per top-of-rack switch and $800K for the cluster switch). A heftier switch like the Mellanox SX6536 gives full cross-bar bandwidth at 56 Gbps and supports up to 648 ports. So take 80 racks with top-of-rack switches as before, but with 22 2RU nodes with dual port 10Gbe links each. We can avoid over-subscribing by having eight 56 Gbps uplinks from each rack, and this would perform a 10 PB shuffle in 46 min.

Observe that the power consumption for movement is about 0.08 to 0.4 watts per node. That is $1/100$th to $1/10$th of what the remaining components of a typical node might consume.

The acceleration we assume is less than $0.5g$. In other projects at IBM, we have electronic components accelerating at $2g$ to $3g$ without having a major impact on reliability. As Table 1 shows, the peak speed of 0.8 m/s (2.9km/hr) is also small. One important concern is resonance, especially with large numbers of nodes. This can be avoided by staggering (in time) the movement of different pairs of adjacent rows.

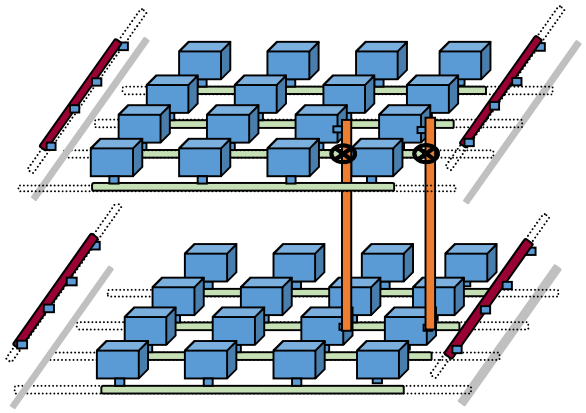

Figure 3: 3-D network.

Impact of friction. Friction is another contributor to power usage. Our peak speed is 0.8 m/s, so the power needed to overcome friction for a 100 g node is $0.1 \times 9.8 \times 0.8 \times \mu$, where μ is the friction coefficient. If we consider only rolling resistance, this is negligible (e.g., for steel balls rolling on steel, μ is about 0.001, leading to 0.8 mW per node). The bearings for the drivetrain might contribute more, and further analysis is needed to see if friction adds significantly to the power usage.

Mechanical issues in 3-D

Drivetrain. In the 2-D version, the drivetrain for moving the nodes along the x and y axes is in the x-y plane. When we go to 3-D, the movement along the z axis would normally collide with the x-y plane drivetrains. So in the z axis movement phase, we need to shift the x and y drive trains laterally. Figure 3 shows 2 levels in a 3-D shuffle. Along the x-y plane, only the drivetrain for row-wise rotation is shown (green for the drives below the nodes, and red for the corner drives that are above the nodes). Along the z axis, only two drivetrains are shown, in orange. Notice that during z-axis movement, both the nodes and the z drivetrain would collide with the x-y drivetrains (see the circles with an X), so the x and y drivetrains must be shifted laterally for this phase. Also, we need to couple the upward and downward movement of opposite drivetrains, to avoid fighting gravity. If this is done with pulleys, the impact of friction could be higher.

Floor load. One limiter of how high we can stack nodes is the node weight, and the pressure it imposes on the floor. Table 1 estimates floor pressure due to the nodes alone. The support structure will definitely add more, but $20\,\text{kg/m}^2$ seems feasible, even with some overhead. The

Table 1: Bandwidth and power calculations for different grid sizes and types of nodes. Floor load ignores structure weight.

	10x10 Grid	100x100 Grid	100 x 100 x 100 Grid
Acceleration $= a$	$4\,\mathrm{m/s^2}$	$4\,\mathrm{m/s^2}$	$4\,\mathrm{m/s^2}$
Node size $= s$, gap $= c$	$s = c = 8\,\mathrm{cm}$	$s = c = 8\,\mathrm{cm}$	$s = c = 8\,\mathrm{cm}$
Peak speed $= \sqrt{a(s+c)}$	$0.8\,\mathrm{m/s}$	$0.8\,\mathrm{m/s}$	$0.8\,\mathrm{m/s}$
Face bandwidth $= b$	$1\,\mathrm{GB/s}$	$1\,\mathrm{GB/s}$	$1\mathrm{GB/s}$
Storage/node $= d$	$100\,\mathrm{GB}$	$100\,\mathrm{GB}$	$100\,\mathrm{GB}$
Total data	$N^2d = 10\,\mathrm{TB}$	$N^2d = 1000\,\mathrm{TB}$	$N^3d = 100\,\mathrm{PB}$
Num. nodes	$N^2 = 100$	$N^2 = 10{,}000$	$N^3 = 1{,}000{,}000$
Mechanical Details			
Movmt Time/step $t = 2\sqrt{(s+c)/a}$	$0.40\,\mathrm{s}$	$0.40\,\mathrm{s}$	$0.40\,\mathrm{s}$
Data transfer time	$2d/b = 200\,\mathrm{s}$	$2d/b = 200\,\mathrm{s}$	$3d/b = 300\mathrm{s}$
Power usage and Floor Load (node mass $m = 100\,\mathrm{g}$ ($500\,\mathrm{g}$))			
Power/node $P = ma(s+c)/2t$	$0.08\,\mathrm{W}$ ($0.4\,\mathrm{W}$)	$0.08\,\mathrm{W}$ ($0.4\,\mathrm{W}$)	$0.08\,\mathrm{W}$ ($0.4\,\mathrm{W}$)
Total Power	$N^2P = 8\,\mathrm{W}$ ($40\,\mathrm{W}$)	$N^2P = 800\,\mathrm{W}$ ($4\,\mathrm{kW}$)	$N^3P = 80\,\mathrm{kW}$ ($400\,\mathrm{kW}$)
Floor Load (kg/m^2)	$m/(s+c)^2 = 4(20)$	$m/(s+c)^2 = 4(20)$	$Nm/(s+c)^2 = 390(1960)$
Bandwidth			
Total time to shuffle	$2d/b + 2Nt = 208\,\mathrm{s}$	$2d/b + 2Nt = 280\,\mathrm{s}$	$3d/b + 3Nt = 420\,\mathrm{s}$
Shuffle Bandwidth	$10\,\mathrm{TB}/208\,\mathrm{s} = 48\mathrm{GBps}$	$1000\,\mathrm{TB}/280\,\mathrm{s} = 3.5\mathrm{TBps}$	$100\,\mathrm{PB}/420\,\mathrm{s} = 238\mathrm{TBps}$

$1960\,\mathrm{kg/m^2}$ seems impractical, so with thick nodes we might be limited to around 25 nodes along the z-axis. As a reference, raised floors in today's data centers have a minimum loading of $150\,\mathrm{lbs/ft^2}$ ($730\,\mathrm{kg/m^2}$) [2].

Buffering Requirement and Skew

Since we use a multi-hop scheme, buffering is an important concern. All the data from nodes on a given row that has to be shuffled onto a given column (or given plane, in 3-D) needs to be buffered at the intermediate node — one that lies on the same row as the source nodes, and the same column (plane) as the targets.

Without data skew, this is exactly the amount of data originally stored on each machine. So we need only a 100% storage buffer. With skew, some nodes may need to buffer much more data. Skew causes other problems as well. Since servers are moving, there is only a limited window within which data can be transmitted. So if there is an imbalance in the amount of data to be transmitted between a pair of nodes, or variance in either data transfer speeds or in the time for which nodes are adjacent, not all of the data can be transmitted before the next node movement. The transmission times are fixed, and are precomputed based on predicted data transfer requirements, with some slack left for data skew.

If due to severe skew data is left over at the end of a shuffle, that is transmitted in a subsequent pass or via a low bandwidth conventional network. Skew can also be addressed by performing transfers through multiple nodes. This approach also means we need not provision extra buffer space to deal with skew.

Heat Dissipation

An important limiter of computer system density is the ability to dissipate heat. Intuitively we believe that moving nodes could actually help heat dissipation. Cooling systems do not uniformly cool the entire room – with moving nodes, no node will be stuck in a hot spot for too long. Also, since all nodes regularly reach the perimeter, nodes that are much hotter than the typical node could be taken out and cooled down separately. Thus the cooling system can be provisioned for the typical rather than worst-case scenarios. But aerodynamics is extremely complex and often counter-intuitive; so this needs careful analysis before we can determine if heat dissipation is better or worse than in conventional systems.

Rail Failures

Single-node failures (other than failure in the wheels or movement mechanism) can be addressed by the same means as in map-reduce applications. Rail failures and failure in the movement mechanism are more serious. The rail structure plays 3 roles: (a) transmitting power to the nodes, (b) supporting the nodes and prevent them from falling, and (c) enabling node movement. In 2-D, we can do (b) via the floor as well. For (a) and (c) one can have backup rails, e.g., one backup rail for the X and one for the Y axis. Rail failure in the 3-D case is much more challenging, because the rails are also responsible for (b), and this needs more investigation.

4 Moving big servers

So far the node form-factor we have considered is a smartphone. But over short distances copper wires give excellent bandwidth at low cost, likely to reach Terabits/s in the near future. Optical communication promises much higher bandwidths over long distances, but this introduces cost and complexity for eletrical-optical-eletrical conversions, circuit switching, etc. So an alternative use case for movement is with bigger nodes.

Consider a 60cm x 60cm x 60cm node that holds 30 TB. Compared to the 8cm x 8cm x 8cm node from Section 3, this uses 420x the volume to hold 300x the data. We want larger nodes because bandwidth out of a node is proportional to its surface area. But if we can get good enough bandwidth via traditional methods within this node, then we can move these big nodes to achieve the long-distance data transfer.

Now consider a 10 PB sort on this configuration. At 30 TB per node, we need an 18x18 grid (2-D). With capacitive coupling, we can get a bidirectional bandwidth of 225 GBps between nodes (a 4cm x 4cm face gets 1 GBps), so the two data transfers contribute 267s to the shuffle time. Movement adds another 40s (for $a = 4\text{m/s}^2$), so the shuffle takes 307s – versus the 46 min achievable with the 648-port switch from Section 3.2. Data transfer time dominates, so using copper (e.g., waveguides) or photonics to transfer across the face will significantly improve our timing.

Moving a big node means we can use a more robust movement infrastructure. First, we can replace the motion rails by a motor and castors on each node, and supply power via brushes. This avoids the rail failure issue.

Second, using big nodes we need much fewer starts and stops. For example in the 100x100 grid of Table 1, each movement step takes 0.4s and each data transfer takes 1s. With a 10x10 grid of larger nodes, each data transfer takes 10s. So there is less pressure to move nodes quicky; we can accelerate 10x slower, and each time nodes stop, we can spend 10x more time making the coupling (and waiting for vibrations to dampen sufficiently), all of which should help with failure rates.

Third, since there are no rails, nodes need not move in lock-step. So we can use any 2-D sorting algorithm, and are not restricted to just our rotation technique. For example, we can do a local sort within every 2x2 quartet of nodes, and then have the nodes move to four quadrants of the system — partitioning the data 4-way, and recurse.

5 Shuffling without stopping

A main challenge with the scheme of Section 3 is the repeated starts and stops of nodes. Besides power consumption, this also adds concerns about vibration and failures. In our next scheme, nodes move at constant speed, except for U-turning at the corner. There are many options for transmitting while in motion, including optical links, capacitive coupling, and mechanisms that rotate a communication element as nodes go by.

The key challenge is how to turn the corner fast enough, to avoid colliding with other nodes. For nodes of size $s \times s$, we use a gap of size $c = s$. This way, by the time the node one position away from the edge reaches the edge, the node at the edge can move away completely and do the turn. This turn involves acceleration along both x and y axes. The discussion below assumes we are in the first phase, where shuffling is done by rotating adjacent rows. A similar procedure applies in the second phase as well.

Acceleration along the perpendicular axis. To move to the next row, we need to travel a distance of $(s+c)/2$ while accelerating, and symmetrically decelerate. So,

$$((s+c)/2) = (1/2)a_2(t/2)^2$$

where a_2 stands for the acceleration along the axis perpendicular to the direction along which the nodes move (when we are rotating adjacent rows, this is the y axis).

Acceleration along the parallel axis. Say nodes are allowed to move past by a distance "slack" at the edges, as they decelerate and reverse direction. Then,

$$2v = a_1 t$$

where v is the speed at which nodes move (until they reach the edge), and a_1 is the acceleration along the direction on which the nodes move. The slack is given by:

$$\text{slack} = v^2/(2a_1)$$

Power consumption. The power needed for cornering has two parts: along the perpendicular axis we need $P_2 = ma_2(s+c)/(2t)$, and along the parallel axis we need $P_1 = ma_1\text{slack}/t$. As Table 2 shows, the consumption reduces sharply compared to the previous scheme. Note that we do not list floor pressure here, because it is identical to Table 1. Friction is still significant, though as in Section 3 the small speed and node weights make us optimistic it will not be a serious problem.

6 Movement in Switched Networks

We now turn to the value of node movement in traditional networks. Consider again the grid of Figure 2. Obviously, the same data shuffle can be done with a 2-D stationary grid with a switch on each row and column (Figure 4 (left)). In principle, this scheme scales; for N^2 nodes we only need switches with bandwidth proportional to N, to get bisection bandwidth proportional to N^2. However, it is hard to pack nodes densely, due to serviceability and cooling considerations.

Table 2: Bandwidth for different grid sizes and different types of nodes.

	10x10 Grid	100x100 Grid	100 x 100 x 100 Grid
Node size $= s$, gap $= c$	$s = c = 8\,\text{cm}$	$s = c = 8\,\text{cm}$	$s = c = 8\,\text{cm}$
Constant speed $= v$	$0.6\,\text{m/s}$	$0.6\,\text{m/s}$	$0.6\,\text{m/s}$
Face bandwidth $= b$	$1\,\text{GB/s}$	$1\,\text{GB/s}$	$1\,\text{GB/s}$
Storage/node $= d$	$100\,\text{GB}$	$100\,\text{GB}$	$100\,\text{GB}$
Total data	$N^2 d = 10\,\text{TB}$	$N^2 d = 1{,}000\,\text{TB}$	$N^3 d = 100\,\text{PB}$
Num. nodes	$N^2 = 100$	$N^2 = 10{,}000$	$N^3 = 1{,}000{,}000$
Time/step $t = (s+c)/v$	$0.27\,\text{s}$	$0.27\,\text{s}$	$0.27\,\text{s}$
Data transfer time	$2d/b = 200\,\text{s}$	$2d/b = 200\,\text{s}$	$3d/b = 300\,\text{s}$
$a_1 = 2v/t$	$4.44\,\text{m/s}^2$	$4.44\,\text{m/s}^2$	$4.44\,\text{m/s}^2$
$a_2 = 4(s+c)/t^2$	$8.77\,\text{m/s}^2$	$8.77\,\text{m/s}^2$	$8.77\,\text{m/s}^2$
slack $= v^2/2a_1$	$4.05\,\text{cm}$	$4.05\,\text{cm}$	$4.05\,\text{cm}$
Power usage (mass $m = 100\,\text{g}$ ($500\,\text{g}$))			
Power/corner $p = ma_2(s+c)/2t + ma_1\text{slack}/t = 0.326\,\text{W}$ ($1.6\,\text{W}$)			
Total Power	$2Np = 6.5\,\text{W}$ ($33\,\text{W}$)	$2Np = 65\,\text{W}$ ($330\,\text{W}$)	$2N^2p = 6.5\,\text{kW}$ ($33\,\text{kW}$)
Total shuffle time	$\max(2d/b, 2Nt) = 200\,\text{s}$	$\max(2d/b, 2Nt) = 200\,\text{s}$	$\max(3d/b, 3Nt) = 300\,\text{s}$

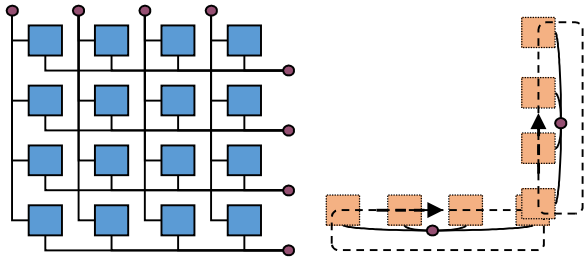

Figure 4: (Left) 2-D network with switches (purple circles) along each row and column. (Right) Drivetrain for moving nodes.

First, we need to leave gaps between nodes to service nodes in the interior of the grid. Second, nodes do not heat up uniformly, and the cooling system does not cool the entire space uniformly. So the cooling system has to be overprovisioned, to keep the biggest gas guzzler, located in the hottest spot, from melting.

So consider a grid of stationary switches as in Figure 4 (left), *but* with the nodes moving. The main challenge is in connecting the moving nodes to the fixed network.

We start with the start-stop movement scheme of Section 3, and add a variation as shown in Figure 4 (right). There is a drivetrain for each row (and column) that provides movement (only shown for bottom row and rightmost column). The slots/frames that hold nodes (shown as orange squares) are always connected to a switch – so the topology is identical to the traditional 2-D network. Note that the vertical drivetrains are on a different plane from the horizontal drivetrains. When a node is moving along a row (or column), it is physically connected to the corresponding slot (via capacitive coupling or otherwise) on the drivetrain for that row (column). But, the slots on the drivetrain are connected via a traditional switch. At the end of a half-rotation, the system transfers to a phase in which nodes travel along the other axis. This involves a "hand-off" in which the node is transferred to the corresponding slot on the drivetrain for the other axis. The hand-offs can be done by electromagnetic solenoids.

Increasing the movement range. During this motion each node repeatedly travels among two rows and two columns: i and $N-i$, (where the rows and columns are indexed from 0 to $N-1$). By making each half-rotation do $N+1$ instead of N steps, we can increase the movement range to span the entire grid.

7 Conclusions

Computer architecture and form factors are continually changing, but one constant principle is to keep processing elements stationary, and move data (and air). We have explored an alternative that pushes sneaker networks to the limit, and moves servers themselves. Our calculations suggest that moving servers is not as hard as it sounds, at least in terms of power consumption and acceleration. At the same time, movement is promising as a way to improve long distance communication bandwidth.

There are many unsolved issues: mechanical aspects, heat dissipation, failure rates, etc. Further, there is a huge industry effort towards high bandwidth optical communication, and volumes may make that far more economical. Still, we hope this paper inspires a few readers to explore alternative designs.

References

[1] A conversation with Jim Gray. *ACM Queue*, 1(4), 2003.

[2] ANSI/TIA/EIA. TIA-942: Telecommunications Infrastructure Standard for Data Centers, 2005.

[3] M. Dobrescu, N. Egi, K. Argyraki, B.-G. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy. Routebricks: exploiting parallelism to scale software routers. In *SOSP*, 2009.

[4] R. B. Garner, W. W. Wilcke, B. J. Rubin, and H. Kahn. Scalable computer system having surface-mounted capacitive couplers for intercommunication. US Patent 7038553 B2, 2006.

[5] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat. B4: Experience with a globally-deployed software defined WAN. In *SIGCOMM*, 2013.

[6] Y. Li, I. Pandis, R. Mueller, V. Raman, and G. Lohman. NUMA-aware algorithms: the case of data shuffling. In *CIDR*, 2013.

[7] A. Vahdat, M. Al-Fares, N. Farrington, R. N. Mysore, G. Porter, and S. Radhakrishnan. Scale-out networking in the data center. *IEEE Micro*, 30(4), 2010.

[8] H. Wang and K. Bergman. Optically interconnected data center architecture for bandwidth intensive energy efficient networking. In *ICTON*, 2012.

[9] W. W. Wilcke, R. B. Garner, C. Fleiner, R. F. Freitas, R. A. Golding, J. S. Glider, D. R. Kenchammana-Hosekote, J. L. Hafner, K. M. Mohiuddin, K. Rao, R. A. Becker-Szendy, T. M. Wong, O. A. Zaki, M. Hernandez, K. R. Fernandez, H. Huels, H. Lenk, K. Smolin, M. Ries, C. Goettert, T. Picunko, B. J. Rubin, H. Kahn, and T. Loo. IBM Intelligent Bricks project—Petabytes and beyond. *IBM Journal of Research and Development*, 50(2.3), 2006.