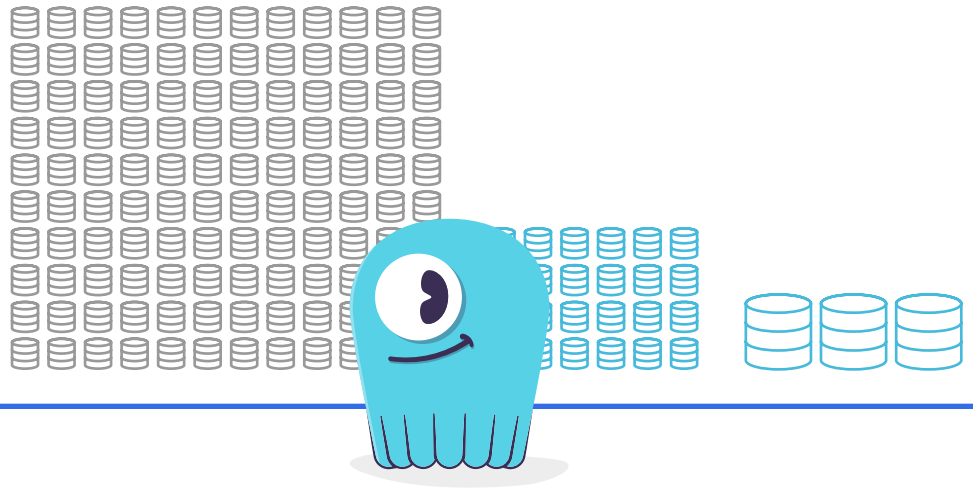
SCYLLA

SCYLLADB WHITEPAPER

Why Scaling Up Beats Scaling Out for NoSQL



CONTENTS

OVERVIEW

With data-intensive applications, scaling out the database is the norm. As a result, deployments are often awash in clusters of small nodes, which bring with them many hidden costs. For example, scaling out usually comes at the cost of resource efficiency, since it can lead to low resource utilization. So why is scaling out so common? Scale out deployment architectures are based on several assumptions that we'll examine in this white paper. We'll demonstrate that these assumptions prove unfounded for database infrastructure that is able to take advantage of the rich computing resources available on large nodes. In the process, we'll show that ScyllaDB, the database for data-intensive apps that require high performance and low latency, is uniquely positioned to leverage the multi-core architecture that modern cloud platforms offer, making it not only possible, but also preferable, to use smaller clusters of large nodes for data-intensive applications.

THE NEW AGE OF NOSQL AND COMMODITY HARDWARE

The NoSQL revolution in database management systems kicked off a decade ago. Since then, organizations of all sizes have benefitted from a key feature that the NoSQL architecture introduced: massive scale using relatively inexpensive commodity hardware. Thanks to this innovation, organizations have been able to deploy architectures that would have been prohibitively expensive and impossible to scale using traditional relational database systems.

Across industries, the ability to align rapidly and efficiently scale applications for growing demand has proven to be a key competitive advantage. As a result, IT groups constantly strive to balance competing demands for higher performance and lower costs. Since NoSQL makes it easy to scale out and back in rapidly on cheaper hardware, it offers the best of both worlds: performance along with cost savings.

Over the same decade, 'commodity hardware' itself has undergone a transformation. However, most modern software doesn't take advantage of modern computing resources. Most data-intensive application frameworks that scale out, don't scale up. They aren't able to take advantage of the resources offered by large nodes, such as the added CPU, memory, and solid-state drives (SSDs), nor can they store large amounts of data on disk efficiently. Managed runtimes, like Java, are further constrained by heap size. Multi-threaded code, with its locking overhead and lack of attention for Non-Uniform Memory Architecture (NUMA), imposes a significant performance penalty against modern hardware architectures.

Software's inability to keep up with hardware advancements has led to the widespread belief that running database infrastructure on many small nodes is the optimal architecture for scaling massive workloads. The alternative, using small clusters of large nodes, is often treated with skepticism.

These assumptions are largely based on experience with NoSQL databases like Apache Cassandra. Cassandra's architecture prevents it from efficiently exploiting modern computing resources—in particular, multi-core CPUs. Even as cloud platforms offer bigger and bigger machine instances, with massive amounts of memory, and ever denser storage options, Cassandra is constrained. It's constrained by many factors, primarily its use of Java, but also its caching model and its lack of an asynchronous architecture. As a result, Cassandra is often deployed on clusters of small instances that run at low-levels of utilization. This low hardware utilization rate results in system sprawl, which equates to operational overhead, with a far larger footprint to keep managed and secure, all of which directly impacts staffing and infrastructure spend and ROI.

ScyllaDB set out to fix these deficiencies with its close-to-the-hardware database design. Written in C++ instead of Java, and optimized to make full utilization of the Linux operating systems found ubiquitously across modern cloud

environments, ScyllaDB implements a number of low-level architectural techniques, such as fully asynchronous shard-per-core. Those techniques, combined with a shared-nothing architecture, enable ScyllaDB to scale linearly with the available resources. This means that ScyllaDB can run massive workloads on smaller clusters of larger, denser nodes – an approach that vastly reduces all of the inputs to total cost of ownership (TCO).

A few common concerns are that large nodes won't be fully utilized, that they have a hard time streaming data when scaling out and, finally, they might have a catastrophic effect on recovery times. ScyllaDB breaks these assumptions, resulting in improved TCO and reduced maintenance.

In light of these concerns, we ran real-world scenarios against ScyllaDB to demonstrate that the skepticism towards big nodes is misplaced. In fact, with ScyllaDB, big nodes are often best for data-intensive applications.

WHAT IS A LARGE NODE?

Before diving in, it's important to understand what we mean by 'node size' in the context of today's cloud hardware. Since Amazon EC2 instances are regularly used to run database infrastructure, we will use their offerings for reference. The following graphic shows the Amazon EC2 I4i family, with increasingly large nodes, or 'instances.' As you can see, the number of virtual CPUs spans from 2 to 128. The amount of memory of those machines grows proportionally, reaching a terabyte with the i4i.32xlarge instance. Similarly, the amount of storage spans from almost half a terabyte to thirty terabytes.

On the surface, the small nodes look inexpensive, but prices are actually consistent across node sizes. No matter how you organize your resources—the number of cores, the amount of memory, and the number of disks—the price to use those resources in the cloud will be roughly the same. Other cloud providers, such as Microsoft Azure and Google Cloud, have similar pricing structures, and therefore a comparable cost benefit to running on bigger nodes.

ADVANTAGES OF LARGE NODES

Now that we've established resource price parity among node sizes, let's examine some of the advantages that large nodes provide.

Instance name	On-Demand hourly rate	vCPU	Memory	Storage	Network performance
i4i.large	$0.172	2	16 GiB	1 x 468 NVMe SSD	Up to 10 Gigabit
i4i.xlarge	$0.343	4	32 GiB	1 x 937 NVMe SSD	Up to 10 Gigabit
i4i.2xlarge	$0.686	8	64 GiB	1 x 1875 NVMe SSD	Up to 12 Gigabit
i4i.4xlarge	$1.373	16	128 GiB	1 x 3750 NVMe SSD	Up to 25 Gigabit
i4i.8xlarge	$2.746	32	256 GiB	2 x 3750 NVMe SSD	18750 Megabit
i4i.16xlarge	$5.491	64	512 GiB	4 x 3750 NVMe SSD	35000 Megabit
i4i.32xlarge	$10.9824	128	1024 GiB	8 x 3750 SSD	75000 Megabit
i4i.metal	$10.982	128	1024 GiB	8 x 3750 NVMe SSD	75000 Megabit

I4i instance specs — Source: AWS

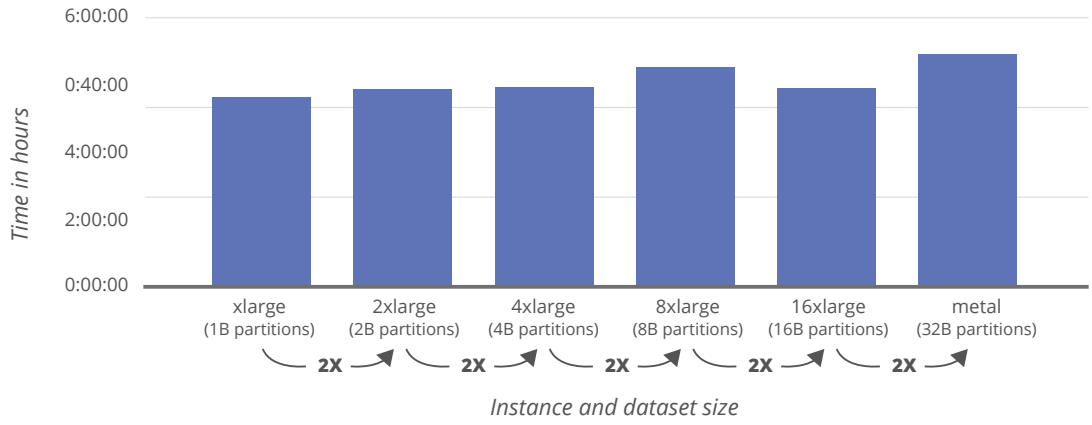
- Less Noisy Neighbors: On cloud platforms multi-tenancy is the norm. A cloud platform is, by definition, based on shared network bandwidth, I/O, memory, storage, and so on. As a result, a deployment of many small nodes is susceptible to the 'noisy neighbor' effect. This effect is experienced when one application or virtual machine consumes more than its fair share of available resources. As nodes increase in size, fewer and fewer resources are shared among tenants. In fact, beyond a certain size your applications are likely to be the only tenant on the physical machines on which your system is deployed. This isolates your system from potential degradation and outages. Large nodes shield your systems from noisy neighbors.
- Fewer Failures: Since large and small nodes fail at roughly the same rate, large nodes deliver a higher mean time between failures, or "MTBF" than small nodes. Failures in the data layer require operator intervention, and restoring a large node requires the same amount of human effort as a small one. In a cluster of a hundred nodes, you'll likely see failures every day. As a result, big clusters of small nodes magnify administrative costs.
- Datacenter Density: Many organizations with on-premises datacenters are seeking to increase density by consolidating servers into fewer, larger boxes with more computing resources per server. Small clusters of large nodes help this process by efficiently consuming denser resources, in turn decreasing energy and operating costs.
- Operational Simplicity: Big clusters of small instances demand more attention, and generate more alerts, than small clusters of large instances. All of those small nodes multiply the effort of real-time monitoring and periodic maintenance, such as rolling upgrades.

In spite of these significant benefits, systems awash in a sea of small instances remain commonplace. In the following sections, we'll run scenarios that investigate the assumptions that lead to the use of small nodes instead of big nodes, including performance, compaction, and streaming. We'll demonstrate that those assumptions don't always hold true when you crunch the numbers.

NODE SIZE AND PERFORMANCE

A key consideration in environment sizing is the performance characteristics of the software stack. As we've pointed out, ScyllaDB linearly scales, enabling it to double performance as resources double. To prove this, we ran a few scenarios against ScyllaDB.

This scenario uses a single cluster of 3 machines from the I4i family. (Although we're using only 3 nodes here, out in the real-world ScyllaDB



Ingestion time remains within bounds as the instance type and dataset size doubles

runs in clusters of 100s of nodes, both big and small. In fact, one of our users runs a 1PB cluster with only 30 nodes). 3 nodes are configured as replicas (with a replication factor of 3). Using a single loader machine, the reference workload inserts 1,000,000,000 partitions to the cluster as quickly as possible. We double resources available to the database at each iteration. We also increase the number of loaders, doubling the number of partitions and the dataset at each step.

As you can see in the chart below, ingest time of 300GB with 4 virtual CPU clusters is roughly equal to 600GB using 8 virtual CPUs, 1.2TB with 16 virtual CPUs, 2.4TB with 16 virtual CPUs and even when the total data size reached almost 10TB, ingest time for the workload remained relatively constant. These tests show that ScyllaDB's linear scale-up characteristics make it advantageous to scale up before scaling out, since you can achieve the same results on fewer machines.

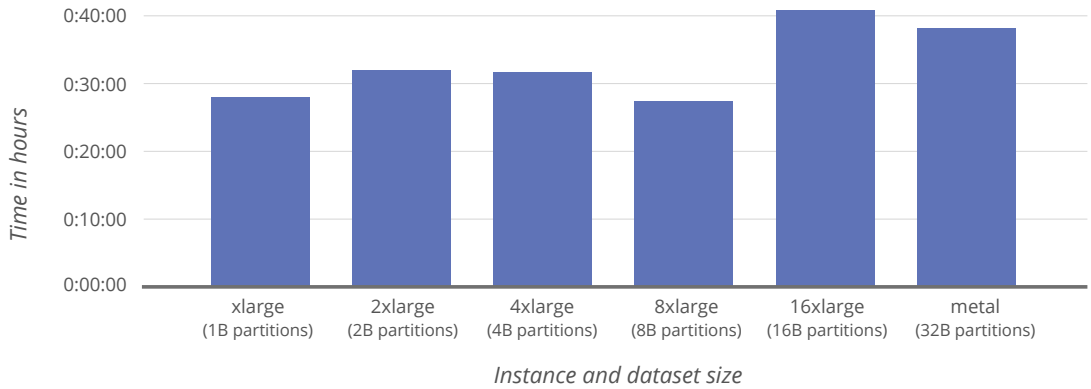
NODE SIZE AND STREAMING

Some architects are concerned that putting more data on fewer nodes increases the risks associated with outages and data loss. You can think of this as the 'Big Basket' problem. It may seem intuitive that storing all of your data on a few large nodes makes them more vulnerable to outages, like putting all of your eggs in one basket. Current thinking in distributed systems tends to agree, arguing that amortizing failures over many tiny and ostensibly inexpensive instances is best. But this doesn't necessarily hold true. ScyllaDB uses a number of innovative techniques to ensure availability while also accelerating recovery from failures, making big nodes both safer and more economical.

Replacing a failed node requires the entire dataset to be replicated to the new node via streaming. On the surface, replacing a node by transferring 1TB seems preferable to transferring 16TB. If larger nodes with more power and faster disks speed up compaction, can they also speed up replication? The most obvious potential bottleneck to streaming is network bandwidth.

But the network isn't the bottleneck for streaming. Even AWS's I4i.4xlarge instances deliver network links up to 25 Gbps—enough bandwidth to transfer 5TB of data within thirty minutes. However, naively streaming data at full-speed may impact the customer-facing workload significantly elevating latencies.

Streaming operations benefit from ScyllaDB's autonomous and auto-tuning capabilities. ScyllaDB queues all requests in the database and then uses a scheduler to execute them in priority order. The schedulers use algorithms derived from control theory to determine the



The time to rebuild a ScyllaDB node remains relatively constant with larger data sets

optimal pace at which to run operations, based on available resources.

These autonomous capabilities result in rapid replication with minimal impact on latency. Since this happens dynamically, added resources automatically factor into scheduling decisions. Nodes can be brought up in a cluster very quickly, further lowering the risk of running smaller clusters with large nodes.

To demonstrate this, we ran a streaming scenario against a ScyllaDB cluster. Using the same clusters as in the previous demonstration, we destroyed the node and rebuilt it from the other two nodes in the cluster. Based on these results, we can see that it takes about the same amount of time to rebuild a node from the other two, even for large datasets. It doesn't matter if the dataset is 300GB or almost 10TB. Our experiment shows that the ScyllaDB database busts the myth that big nodes result in big problems when a node goes down.

Calculating the real cost of failures involves more factors than the time to replicate via streaming. If the software stack scales properly, as it does with ScyllaDB, the cost to restore a large node is the same as for a small one. Since the mean time between failures tends to remain constant regardless of node size, fewer nodes also means fewer failures, less complexity, and lower maintenance overhead.

Efficient streaming enables nodes to be rebuilt or scaled out with minimal impact. But a second problem, cold caches, can also impact restarts. What happens when a node is restarted, or a new node is added to the cluster during rolling upgrades or following hardware outages? The typical database simply routes requests randomly across replicas in the cluster. Nodes with cold caches cannot sustain the same throughput with the same latency.

ScyllaDB takes a different approach. Our heat-weighted load balancing technology, intelligently routes requests across the cluster to ensure that node restarts don't impact throughput or latency. Heat-weighted load balancing implements an algorithm that optimizes the ratio between a node's cache hit rate and the proportion of requests it receives. Over time, the node serving as request coordinator sends a smaller proportion of reads to the nodes that were up, while sending more reads to the restored node. The feature enables operators to restore or add nodes to clusters efficiently, since rebooting individual nodes no longer affects the cluster's read latency.

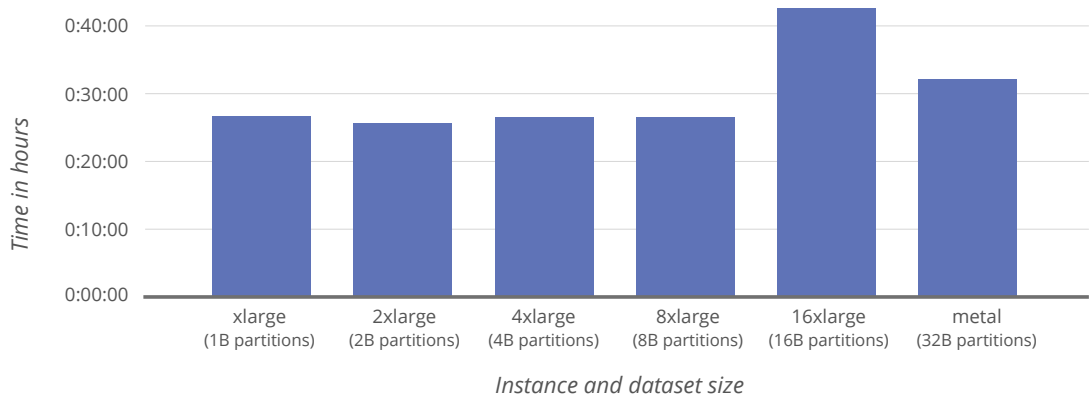
NODE SIZE AND COMPACTION

Readers with real-world experience with Apache Cassandra might react to large nodes with a healthy dose of skepticism. In Apache Cassandra, a node with just 5TB can take a long time to compact. (Compaction is a background process similar to garbage collection in the Java programming language, except that it works with disk data, rather than objects in memory).

Since compaction time increases along with the size of the dataset, it's often cited as a reason to avoid big nodes for data infrastructure. If compactions happen too quickly, foreground workloads are destroyed because all of the CPU and disk resources are soaked up. If compaction is too slow, then reads suffer. A database like Cassandra requires operators to find the proper settings through trial and error, and once set, they are static. With those databases, resource consumption remains fixed regardless of shifting workloads.

As noted in the context of streaming, ScyllaDB takes a very different approach, providing autonomous operations and auto-tuning that give priority to foreground operations over background tasks like compaction—even as workloads fluctuate. The end result is rapid compaction with no measurable effect on end user queries.

To prove this point, we forced a major compaction on a node member of the same cluster as in the previous experiment.



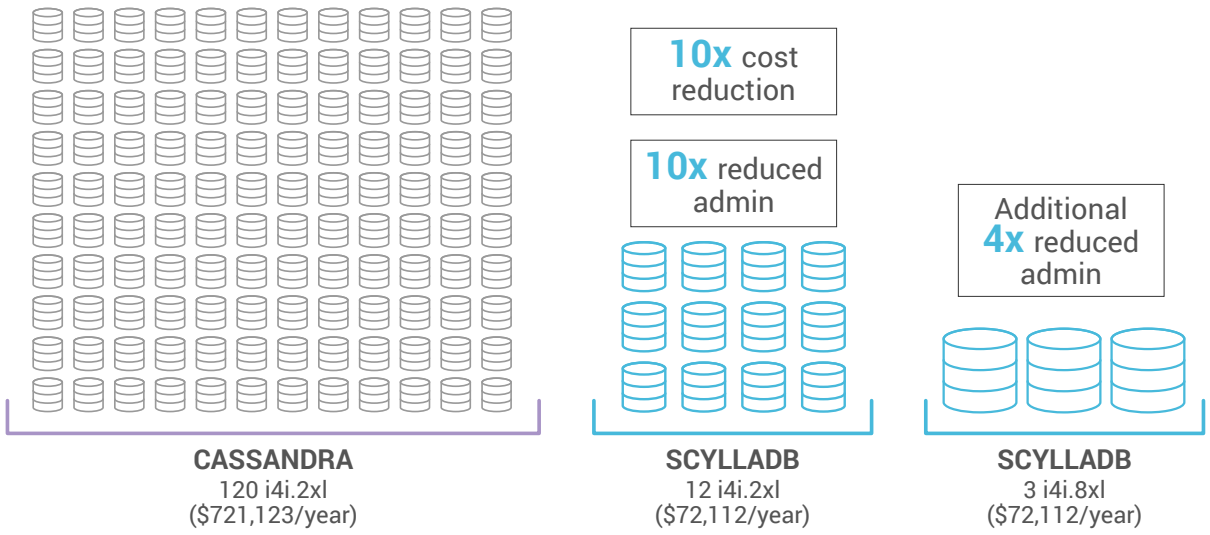
Compaction time in ScyllaDB under different dataset sizes

These results show the time it takes to run a major compaction over a growing dataset, and how it completes in a matter of minutes. This demonstrates that one of the primary arguments against using large nodes for data-intensive applications hinges on the false assumption that compaction of huge datasets must necessarily be slow. ScyllaDB's linear scale-up characteristics show that the opposite is true; compaction of large datasets can also benefit from the resources of large nodes.

THE DATABASE MAKES A DIFFERENCE

As our scenarios demonstrate, running on fewer nodes is inherently advantageous. The highly performant ScyllaDB database reduces the number of nodes required for data-intensive applications. Its scale-up capabilities enable organizations to further reduce node count by moving to larger machines. By increasing the size of nodes and reducing their number, IT organizations can recognize benefits at all levels, especially in total cost of ownership (TCO). Moving from a large cluster of small nodes to a small cluster of large nodes has a significant impact on TCO, while vastly reducing the mean-time between failures (MTBF) and administrative overhead.

The benefits of scaling up that we have demonstrated in this paper have been proven in the field. The diagram below illustrates the server costs and administration benefits experienced by a ScyllaDB customer. This customer migrated a Cassandra installation,



ScyllaDB reduced server cost by 10X and improves MTBF by 40X (120 nodes to 3)

distributed across 120 i4i.2xlarge AWS instances with 8 virtual CPUs each, to ScyllaDB. Using the same node sizes, ScyllaDB achieved the customer's performance targets with a much smaller cluster of only 12 nodes. The initial reduction in node sprawl produced a 10X reduction in server costs, from $721,123 to $72,112 annually. It also achieved a 10X reduction in administrative overhead, encompassing failures, upgrades, monitoring, etc.

Given ScyllaDB's scale-up capabilities, the customer then moved to the larger nodes, the i4i.8xlarge instance with 32 virtual CPUs each. While the cost remained the same, those 3 large nodes were capable of maintaining the customer's SLAs. Scaling up resulted in reducing complexity (and administration, failures, etc.) by a factor of 40 compared with where the customer began (by moving from 120 nodes to 3).

As we've seen, there are many advantages to scaling up a distributed database environment before scaling out. If you've already scaled out, scaling up offers an escape from the costs and overhead of node sprawl.

ScyllaDB gives users both the scale out benefits of NoSQL along with scale up characteristics that enable database consolidation and improved TCO.

Ask yourself...

- Are you concerned your database isn't efficiently utilizing all of its CPUs and disks?
- Are you investing too much time in managing excessively large clusters?
- Does your team have to replace dead nodes at odd hours on the weekends and holidays?
- Are you struggling to scale with your business as you grow 2X, 4X, 10X?
- Are you unable to meet service level agreements during failures and maintenance operations?

If the answer to even one of these questions is 'yes," we'd suggest evaluating your distributed database options. Open source ScyllaDB is available for download from our website. Or start by running in minutes on ScyllaDB Cloud, our fully-managed database-as-a-service.

NEXT STEPS

Get started with ScyllaDB

Learn more at ScyllaDB University

Explore papers, videos, benchmarks & more

ABOUT SCYLLADB

ScyllaDB is the database for data-intensive apps that require high throughput and predictable low latency. It enables teams to harness the ever-increasing computing power of modern infrastructures – eliminating barriers to scale as data grows. Unlike any other database, ScyllaDB is built with deep architectural advancements that enable exceptional end-user experiences at radically lower costs. Over 400 game-changing companies like Disney+ Hotstar, Expedia, FireEye, Discord, Crypto.com, Zillow, Starbucks, Comcast, and Samsung use ScyllaDB for their toughest database challenges. ScyllaDB is available as free open source software, a fully-supported enterprise product, and a fully managed service on multiple cloud providers. For more information: ScyllaDB.com

SCYLLADB.COM

SCYLLA

United States Headquarters
1309 S Mary Ave
Sunnyvale, CA 94087 U.S.A.
Email: info@scylladb.com

Israel Headquarters
11 Galgalei Haplada
Herzelia, Israel