Pyramid Scaling Guide

Guidelines and best practices for how to scale Pyramid

Contents

Overview

Scaling an analytics platform can be complicated due to the many variables and factors that affect the performance of the "analytics stack". These factors include hardware choices; networking; the host operating system; the client browser; the underlying data stack technologies; and the data footprint itself. Other key variables relate to activity volumes - which is a combination of concurrent user patterns; typical query sizes / complexity; and content design.

Owing to this complexity, there is no set formula that can be easily applied to every deployment scenario, beyond the technical steps needed to create a scaled-out instance. Instead, there are a variety of best practices and ideas on how to create scale and ways to measure what factors in the stack need attention.

The good news is that Pyramid was designed to scale up and out to meet those needs. However, you will need to consider other aspects of your deployment in arriving at the right solution. This guide is designed to help you come to the right decisions.

The first step is understanding how Pyramid itself can be installed.

Steps for Scaling Pyramid

The following explains different options for how Pyramid can be deployed. An explanation of the server tiers in Pyramid is first required to better understand these options.

Pyramid Server Tiers

Pyramid is made up of a set of server-side Java service application tiers that work together to respond to user requests and analytic tasks. The tiers are broken in 4 core servers; 2 peripheral servers, 3 optional servers and the Pulse server.

Core Servers

- **Web Server**: This tier hosts the client application and provides the main entry and exit point for requests and responses to both the client tools and API calls.

Resource usage is medium low.

- **Router Server**: This tier provides the message routing decisions required in a multi-node deployment.

Resource usage is low.

- **Runtime Server**: This tier processes all real-time requests and tasks like querying, content management and administrative functionality.

Resource usage is high, and speed is key.

- **Task Server**: This tier processes all batch and off-line requests and tasks such as data preparation/ETL jobs, publications, log cleaning and provisioning.
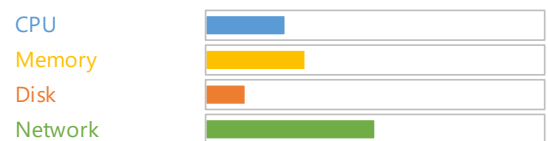
Resource usage is high, but speed it less key.

- **Augmented Analytics Server:** This tier (also known as the "AI" server) processes all AI and machine learning requests related to Python, R and NLQ.

Resource usage is high, and speed is key.

CPU
Memory
Disk
Network

Peripheral Servers

Peripheral servers cannot be installed separately and are installed automatically as companion services with the core tiers.

- **File Server**: This service is installed on each machine hosting Pyramid and facilitates all file and data movements between servers in a multi-node deployment.

Resource usage is low.

CPU
Memory
Disk
Network

- **Agent**: This service is installed on each machine hosting Pyramid and monitors the health of all Pyramid services installed.
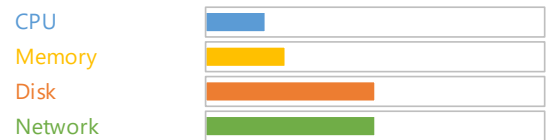
Resource usage is very low.

CPU
Memory
Disk
Network
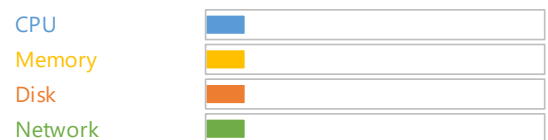
Optional Services

- **Windows Connector Service**: this optional tier is needed when customers plan to query Microsoft OLAP or Tabular data sources. This tier is not required if these data sources are not used.

Resource usage is high, and speed is key.

CPU
Memory
Disk
Network

- **In-memory Server**: this optional service is designed to house and run Pyramid's in-memory database server. This tier is not required if the in-memory engine is not used[1].

Resource usage is high, and speed is key.

CPU
Memory
Disk
Network

- **Repository Database Server**: This optional server is used to house the database repository holding all the meta structures used in the platform. The functionality itself is NOT optional. However, customers can choose to use the internal PostgreSQL database engine or use a 'remote' database server to house the repository (PostgreSQL, MS SQL, Oracle).

Resource usage can be medium to high.

CPU
Memory
Disk
Network

---

[1] The geospatial mapping tools in the application use a special database housed in the in-memory engine. If it's not installed, the mapping engine will not be available.

Pulse

- **Pyramid Pulse Server**: this tier is NOT part of the standard install – and is by design installed on separate remote hardware to connect remote databases to the main Pyramid installation. Scaling the Pulse server is covered separately below.

Resource usage is high, and speed is key.

CPU
Memory
Disk
Network

Basic Scale Models

Since each tier of Pyramid can be installed on one or more machines in almost any combination, the scale out permutations are almost endless. The following 3 basic models shown in figure 1 below cover the skeletal approaches.

Note that the use of firewalls and separate data stack machines is shown in a highly simplistic flow below. They may not be appropriate in all situations and the network configurations may be far more complex.
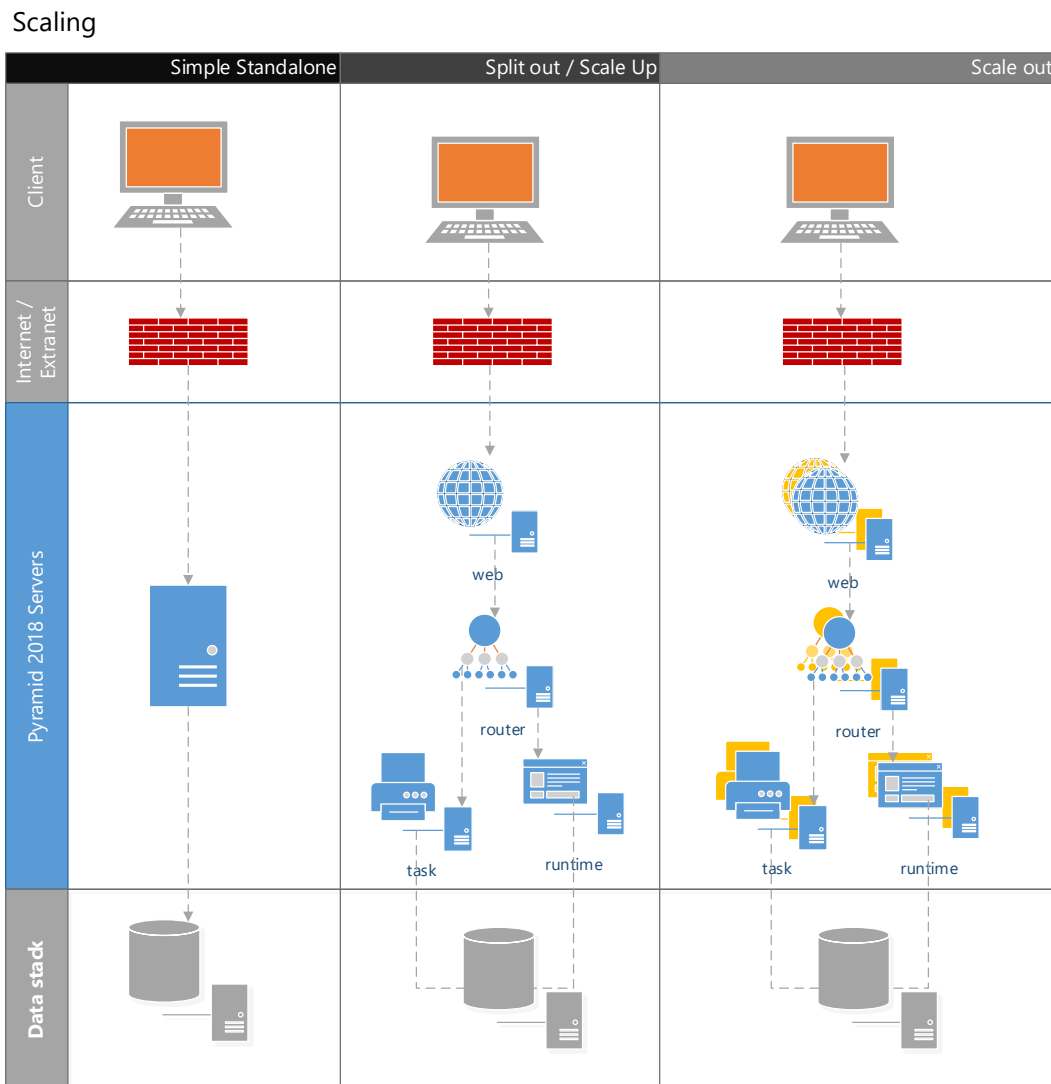
Scaling

Figure 1 Basic Scale Models

Simple Standalone

The simplest is to install all Pyramid tiers on a single machine. If the host server is big enough[2] this approach is the least complicated and fastest deployment model. This approach includes all the core, peripheral and optional servers installed on one machine.

If resourced well, Pyramid will comfortably use all the server's resources and scale up as needed. Critically, the data stacks that are being used for analyses should be on separate hardware unless they are small, or the hardware housing Pyramid can accommodate its load as well.

Split Out / Scale Up

This type of model involves splitting out and installing each of Pyramid 4 core tiers and 3 optional tiers on separate machines – creating a multi-node cluster. This includes variations where 2 or more tiers share hardware in varying combinations. This approach allows customers to put resource-intensive tiers on their own hardware – and therefore provides a more efficient use of given resources. It also offers better scale up options for any given tier/server without wasting those same resources on the low-demand tiers (e.g. if the solution needs more processing for real-time querying, the Runtime server can be allocated more CPU's or memory in a virtualized environment). The 4 tiers most relevant here being the Runtime, Task, Windows Connector and In-memory servers. Putting the web and router servers on the same machine is a typical approach, with other tiers on their own hardware. In addition, using a remote database server to house the repository is another option.

The data stacks that are being used for analyses are usually on separate hardware - in much the same way that each Pyramid tier is separated out.

Scale Out

The scale out model is a variation of the split out, except that it involves adding one or more duplicate tiers on multiple nodes into the cluster. This approach involves adding multiple servers to unblock bottlenecks on any given tier. So, for example, if heavy batch processing is required, customers can deploy 2 or more task servers while only using 1 web, runtime and router server.

While the scale out model does not preclude scaling up hardware for any tier, it does facilitate failover and high availability and the use of different physical servers for scaling. There is also evidence that scaling out produces slightly better results than the same resources used for scaling up.

Again, a remote database server is typically used to house the repository and the data stacks that are being used for analyses are on separate hardware.

Tier Communication

In the split out and scale out models, the Router server handles all communication flows and ensures that all data packets move around the different tiers efficiently and effectively.

---

[2] "Big" implies something strong enough to house the entire Pyramid application and provide enough processing resources for querying, batch processing and the running of all other peripheral services and the host operating system. As described in this document each deployment is unique. So, there is no one definition of what that is.

Multi-node options

Every tier in the Pyramid stack can be installed multiple times in the cluster. The following considerations should be noted in a multi-node environment:

- For core tiers:
  - The Router server acts in an active-passive, failover model. This means, one router is the master, and the secondary routers are passive until the primary fails.
  - The web servers can act in an active-active, load balance model. However, the customer needs to use their own load balancing technologies as part of their web farm strategy.
  - The runtime and task servers act in an active-active, load balance model. This means jobs are shared between all tiers all the time. The Router server handles the load balancing.
- For Optional Tiers
  - The Windows connector acts in an active-active, load balanced model. The Router server handles the load balancing.
  - The in-memory engine is only available as a single node installation. This means you can have multiple in-memory servers hosting multiple databases and models. However, they cannot be load balanced.
  - The database repository clustering options are based on the underlying technologies. PostgreSQL, Microsoft SQL and Oracle all offer multi-node clustering options. However, the setup and maintenance of these high-availability systems is the responsibility of the customer.

Multi-node Installation Procedures

Basic instructions for installation can be found in Pyramid online help.

To create a multi-node installation of Pyramid, use one of the advanced installers (Windows or Linux) and choose the "multi-server" option. From the tier list, select which components you wish to install.

The initial installation, regardless of tier type, must include the creation and initialization of the database repository (choose "new"). All subsequent node installations must then point to that repository (choose "current"). By using the common repository, we are simply adding the new node to the existing installation cluster. Once added, the cluster will self-recognize the components and nodes and use them accordingly.

MVP: Minimum Viable Product

When the minimum requisite tiers are available in the cluster, "MVP" status is achieved, and the product can be initialized and become partially functional. **MVP is achieved when at least 1 web, 1 runtime and 1 router tier is added to the cluster (on one or more nodes)**. It is highly recommended to add a task server as well, and to add the router last in the installation sequence.

Once the application is initialized users can start working with the product. Post MVP, new nodes can easily be added to the cluster by installing them as needed and using the common repository instance.

Removing Nodes

Uninstalling the software from a server will remove the node from the cluster. However, care should be taken that the MVP tiers remain intact. If they are missing, the cluster will fail.

Scaling Considerations

Before you can pick a scaling strategy, you need to consider all the variables that can affect the approach. As mentioned, this includes hardware choices and networking; the host operating system; the client browser; the underlying data stack technologies; and the data footprint itself. The other key variables relate to activity volumes which is a combination of concurrent user patterns; typical query sizes / complexity; and content design requirements.

Hardware

General Resources

Data analytics is a resource intensive domain, and requires numerous calculations and CPU cycles; plus it also needs disk space to handle the vast volumes of data. Often, resources allocated to the environment are extremely understated.

- **Memory** is used heavily by Pyramid to improve performance and processing. This impacts the runtime, task, windows connector and in-memory tiers. Providing adequate memory allows bigger jobs to run without disk paging. Memory is less relevant for the router and web tiers.
- **CPU** is used heavily by Pyramid for processing. Like memory, this impacts the runtime, task, windows connector and in-memory tiers. Providing adequate CPUs allows more jobs to run concurrently without queueing. CPUs are less relevant for the router and web tiers. A key aspect to note is that a given task must run on a single CPU thread (or core) at any time. So, for example, only 16 activities can run on 16 cores at a given time. If there are more jobs, the CPUs merely "thrash" and "switch" activities and it seldom produces good performance.
- **Disk** is not crucial for most activities in Pyramid. However, it is used as the storage medium for the database repository and can heavily affect its performance. Disk is also used for data preparation/ETL jobs when processing large files. The read/write performance can be affected by slow disks for these types of operation.

Review the online help for System Requirements and minimum recommended CPUs and RAM

Virtualization

Generally, virtual machines and bare metal machines will operate with performance profiles expected from those technologies in typical conditions. However, over allocation of CPUs and memory in virtualized deployments will severely impact performance (as described above).

Networking

As Pyramid is usually connecting to networked resources (both intranet and extranet), networking can have an impact on performance. Server-side tiers should be installed on the same network segment (in the same domain), with fast NICs and switches. The data stacks that will be analyzed by Pyramid should follow suit. Where possible, Pyramid should be installed on the same network segment as the data stack to improve performance.

Client connectivity is also affected by networking speed. As part of the Pyramid design much emphasis was placed on reducing the networking requirements in Pyramid's client-server communication design. Nevertheless, network latency is something that will affect the perceived speed of the application.

Host Operating System

Both the Windows and Linux builds of Pyramid can scale up resource usage as needed. In some operations, the Linux version is shown to be slightly faster at processing the Java code base. However, the In-memory engine performs slightly better on Windows.

Care should be taken that in both host systems, all non-essential competing processes are minimized to allow Pyramid to fully utilize all resources. Good examples include performance logging, anti-virus applications and other competing applications.

Client Browser

The choice of client browser has a relatively large impact on performance owing to the difference in browser technologies and engines when processing the HTML5 content. Generally, Chrome and Firefox provide the fastest experience for users, followed by Opera. Other client browser technologies such as Internet Explorer, Edge and Safari support HTML5, however performance is slower.

In the event a customer cannot deploy a fully compatible HTML5 browser, the Pyramid Desktop client provides a convenient alternative and involves a small installation process.

Data Stack

The database engine that will be analyzed by Pyramid **has a significant impact on the performance and scalability** of the platform. While it may be convenient to connect directly to various stacks, they may not be properly designed for the speed of data analytics or properly configured for such operations. As such, care should be taken on how data stacks are prepared for analysis and expectations matched accordingly. Explaining how to approach this item is beyond the scope of this document, however, some general ideas are provided below:

- **Relational databases** are generally weak for high-speed, high-concurrency analytic queries. If used, they need to be properly optimized with file partitioning, column indexing and table structure. Relational engines are also very sensitive to hardware choices – especially disk speed.
- **In-memory databases** perform extremely well for high-speed analytic queries. However, they are very sensitive to CPU resources and are limited by available memory. If memory is sufficient, concurrency is heavily affected by the number of available core CPUs to process requests.
- **OLAP databases** have a similar profile to in-memory systems, however, they often store results on disk as well. This means they are less reliant on memory but are more affected by disk speed and CPUs.
- **MPP databases** are high-end solutions designed for high-speed, high-concurrency queries. However, the proper optimization steps for each technology should still be followed.

Data Footprint

Obviously, the larger the data footprint, the bigger the impact on performance. This is further impacted by the type of data stack technology (described above). Pyramid makes querying databases easy for end-users, which allows them to consume enormous amounts of detailed data through the engine. When the data footprint is extremely large, the possibilities for performance issues can increase. Often, the best approach is to put an end-user education program in place to ensure analytical best practices are adopted and the systems are not misused.

The design of the data footprint has a big impact on performance. Often, this is unavoidable because it reflects business requirements. The variations and techniques for solving performance problems across all data source types are beyond the scope of this document. However, the following tips are applicable to many relational database engines.

Depth vs Breadth

Usually, the breadth of a database (number of tables, joins and number/type of columns) can have a bigger impact on performance than the depth (number of rows).

For relational technologies, 'depth' issues are usually solved with file structures, table partitioning and disk speed. 'Breadth' issues, on the other hand, require more attention and usually involve structural changes to the database.

Here are some general tips for handling breadth issues on structured relational data sources:

- Reduce the number of table joins needed to query the system. Effectively, this means trying to keep transactions to a single fact table, with a simple, single inner join to all peripheral dimensional tables.

- Where possible ensure dimensional tables are denormalized down to single tables rather than chains of tables. The use of star schemas or snowflake schemas (The snowflake schema is a variation of the star schema, featuring normalization of dimension tables) are a better approach.
- Instead of using columns from the fact table for dimensional attributes, create dimensional tables or indexed views BEFORE adding them to the model schema. The cost of finding unique values and keys from deep fact tables for each query can be a very expensive and will heavily affect performance.
- Index all primary and foreign key columns. Also, relational databases often perform better with numeric keys.
- Inner joins are generally the cheapest joins to create and process. So, try and ensure keys are a perfect match between tables.
- Reduce or remove all "blob" columns from tables that will be used for analytics. Due to their size, they can impede read speed. Instead, put them in adjunct tables that can be drawn in if needed.

Activity Volumes

The amount and size of activities planned for the system have a profound impact on the scaling strategy. Activities are classified as all events triggered in the system in each period. This means user concurrency as well as the types of queries the users will run are the key aspects to measuring the size and volume of activity.

User Concurrency

The number of expected concurrent users on the system has a tremendous impact on performance. Each user's interactions with the system produces events and transactions that mostly need to be processed on the servers, since the client application is delivered through a thin, JavaScript web application in the user's browser.

Like all client-server designs, Pyramid was architected to comfortably handle concurrent requests from multiple users at the same time in a fast, efficient manner. But there are limits to everything. The scaling options are designed to give customers the best opportunity to increase those limits as needed.

Measuring Concurrency

The hardest issue to resolve with concurrency relates to determining the expected usage of the product expected over time. Usually, this is evaluated for a given period: daily, weekly or monthly. The simplest approach is to decide that every user in the system could theoretically access it at the same time. However, this is usually a gross overestimation of the figures, and can lead to expensive hardware resourcing.

Many enterprise applications are rated on a 10% concurrency rate for a given user population. Pyramid has no simple answer to this question and cannot therefore make a specific recommendation on scale out formulations. Instead, our approach is to start with a reasonable and modest set of concurrency assumptions; measure usage over time; and continue resourcing the cluster as needed from there.

There are numerous tools for measuring actual concurrency. However, the transaction and audit logs in Pyramid itself provide the most accurate way of measuring both the number of concurrent users as well as the number of concurrent querying events in the system.

Peak vs Average

Once usage is better understood, the next key step is to decide if the system's performance should be scaled out for peak periods or to tolerate some slowness and instead cater for average usage periods (with lower resourcing). The platform's design is centered on the idea that it is easy to keep adding or modifying software and hardware resources in the system, so one approach does not preclude the other.

Query Size and Complexity

While the number of concurrent users is a key factor in determining the scale strategies, an equally important consideration is the types of queries users will run. While user usage patterns can be become more and more predictable over time, query

patterns can remain elusive. As such, a strong transaction monitoring regime using Pyramid's logs is key to refining query patterns.

Query Size

Regardless of the pattern, queries affect performance as follows:

- Normal queries (up to 500 rows by 4-8 columns) are highly scalable, because the data stack, Pyramid servers and Pyramid client can process and display results quickly and efficiently. Most analysts would admit that queries exceeding this size are generally not analytical. Rather they are data fishing exercises or data extraction processes.
- Bigger queries (up to 5000 rows by 4-8 columns) are still scalable, although they are generally useless analytically unless they are viewed in 'data reduction' visualizations (scatter, tree map etc.).
- Large queries (10,000+ rows by 8+ columns) require more processing time by both the data stack and Pyramid – and will consume more network bandwidth. Client browsers will also be impacted in some situations when rendering results. These queries are either built by mistake or are used for data extraction.
- Mega queries (100,000+ rows by 8+ columns) should be used rarely and will impact all tiers in the stack

Therefore, if queries are normal to big, scale out strategies will be less concerned with query size. If querying is on the larger side, than the scaling strategy must take this into account (especially the data stack).

Query Complexity

The complexity of a query is driven by the several elements and can vary greatly between data technologies. The following broadly applies to all data stacks:

- Number of joins needed in the SQL statement. The more joins, the slower the query response, and the greater the data stack will need to work to respond. Joining has a greater impact on some of the newer data engines like RedShift, BigQuery, Presto and Drill. Keeping the data model schema simpler can reduce this issue.
- Calculations impact performance. The heavy use of custom calculations (members or measures) will affect performance in both MDX and SQL.
- MDX queries are affected by detail or grain, and the level of aggregation that may be available in the cube. Tabular results are not generally affected by grain. However, they are much more sensitive to calculations and dimension nesting. If granular querying is required, consider using an alternative engine to SSAS OLAP cubes.

Content Design

Performance is basically a function of the number of users (a) multiplied by the number of queries (b) multiplied by query size and complexity of a queries (c).

Content design (b) dictates the number of queries to be executed in each analysis. While most of the content design decision is driven by business requirements that are generally inflexible, it is worth understanding their impact.

Dashboards and Publications

When users build dashboards, they typically include 2 or more queries on the visible canvas, and these are usually accompanied by filters (or slicers). When launched, the engine needs to draw all these elements concurrently and it therefore affects performance. Reducing the number of visible elements on a canvas lightens the load.

Filter and Slicers

Most users do not realize that filters (slicers) are also generated by queries to the same data source and require processing and rendering. Further, slicer lists can become exceedingly large (in the thousands of elements). Reducing slicers or limiting their element size will also lighten the load.

Cascading

When slicers are cascaded (x affects y, y affects z; etc.), they need to be executed in order and then injected into the final query. This prevents asynchronous processing and can slow execution times. Ironically on the other hand, cascades lighten

the load because they operate as a filtering mechanism for downstream elements, reducing the amount of excess data to retrieve and draw in the client.

Non-Query based Activities

One of the greatest impacts on performance may come from peripheral activities executed on the platform outside of interactive queries and analytics. This includes, but is no limited to, data modeling, publication batch processing, provisioning, and system maintenance.

Data Modeling

The process of designing and processing data models can heavily impact the resources of the system. The runtime engine server is used heavily during the model design process (for example with 'previews'), while the task engine is used to process the model (scheduled or otherwise). The number of active data modeling jobs and concurrent users designing models should be accounted for in sizing the number of both runtime and task engine servers.

Publications

Publication execution is always run from the task engine, even in preview mode. So, it affects the performance and resources of the runtime engine less. Nevertheless, it does impact the system.

Provisioning and System Maintenance

These background administrative tasks are all launched from the task engine servers. So they only lightly affect the performance and resources of the runtime engine.

See comments on real-time and batch processing for more.

Scaling Strategies

Using our basic model options described above and an understanding of the key variables affecting performance, we can now consider several scaling strategies.

Techniques

Pyramid recommends 2 techniques for scaling out the solution: proactive 'prototyping', or reactive 'increments'. These techniques can be used together – they are not mutually exclusive.

Prototype Scaling

In this approach, the environment administrators would take some baseline hardware footprint, which represents a rough estimate of the footprint needed to support users. Based on the statistics, administrators can then formulate the scale to meet real-world needs by scaling out the baseline.

Using load testing tools like Microsoft Visual Studio or JMeter, administrators would simulate several "sessions" that users may follow in their use of the product. Using these tools, administrators can determine if they have enough hardware and the right combination of stack elements to meet the user volumes and activities expected on the system with the baseline hardware chosen.

While this approach is proactive, and allows administrators to put the right solution in place before deployment, it does require realistic tests to be created with realistic user concurrency rates.

Incremental Build Out

In this approach, the environment administrators would put in place a reasonable amount of hardware which represents a rough estimate of the footprint needed to support the real-world users.

Using transaction logs and other performance monitoring tools, administrators can then modulate the solution based on review of performance. At the start, the review would be quite frequent. As the solution matures, the reviews can be sparser. The key here is to readily add new nodes to the cluster to meet performance demands without long delays.

While this approach is only reactive, it allows administrators to put the right solution in place based on real world outcomes, rather than guesses made using a simulator.

Guidance

Below are some ideas on deployments based on size. These are ONLY CONSERVATIVE GUIDES. To get accurate scaling strategies customers need to use the insights from this document and the techniques described above to tune the solution to their own needs. Pyramid's baseline scaling results are presented for illustrative purposes in the Appendix.

Note that concurrency in the guides below is assumed to be at 25%, rather than 10%. This figure, as described above, needs to be measured in each deployment context.

Bare Minimum Server Specification

The "bare" minimum requirement – and we emphasize "**BARE**" - for a given Pyramid designated machine is at least 8 cores and 8 to 16Gb of Ram. Disk space varies, but at least 2Gb of space is required to install the entire Pyramid application on a single server with all the 3rd party drivers and connectors (included in the build). Note that there are no such limits on machines hosting data stack technologies or the database repository.

Regardless of these requirements, each machine described below should be sized according to the techniques described above.

Single and Dual Machine Deployments

Except in the smallest and quickest of scenarios, a single machine installation is generally inadequate. Pyramid was designed to scale and has an architectural footprint for high volume and not minimalist deployments. However, it may be a reasonable place to start before adding more nodes to your cluster. Single machine deployments are good for POC's and small workgroups. Even so, single machines that need to respond to queries and process large batch jobs in the background can be insufficient in real-world production.

Dual server deployments are obviously better than single machine installations. If there are only 2 servers available, a good split of tiers for real-time querying performance is:

- One machine for the Web server, Router server, Task server and repository database
- One machine for the Runtime server

For a batch oriented solution swap the task and runtime engines. The data stack, to be analyzed, should be separate, including the In-memory database.

The following scenarios are provided as guidelines and the performance will be dependent on a number of environment variables. The deployment scenarios focus on scaling up a number of machines, but equally increasing machine capacity rather the number of machines is an alternative approach.

Small Deployments

If you are planning to deploy 100 users or less, with 25% concurrency, reasonable queries and a reasonable data footprint, Pyramid recommends starting with a multi-node cluster of 2-4 machines:

- One machine for the Web and Router servers
- One machine for the Runtime server
- One machine for the Task server
- One machine for the repository database
- The data stack, to be analyzed, should be separate, including the In-memory database.

Medium Deployments

If you are planning to deploy 200 users or less, with 25% concurrency, reasonable queries and a reasonable data footprint, Pyramid recommends starting with a multi-node cluster of 5-6 machines:

- One machine for the Web server
- Optionally one machine for the Router server (or combine with the web server)
- Two machines for the Runtime server
- One machine for the Task server
- One machine for the repository database
- The data stack, to be analyzed, should be separate, including the In-memory database.

Large Deployments

If you are planning to deploy 500 users or less, with 25% concurrency, reasonable queries and a reasonable data footprint, Pyramid recommends starting with a multi-node cluster of 8-9 machines:

- Two machines for the Web server (the customer will need to use their own web-farm load balancers)
- One machine for the Router server
- Three machines for the Runtime server
- Two machines for the Task server (this is key if there are many publications and ETL processes)
- One machine for the repository database

- The data stack, to be analyzed, should be separate, including the In-memory database. Depending on data stack technology, these machines should be scaled out too.

Enterprise Deployments

If you are planning to deploy thousands of users, with 25% concurrency, reasonable queries and a reasonable data footprint, Pyramid recommends starting with a multi-node cluster where each tier is **at least 2 machines per tier**:

- Two or more machines for the Web server (the customer will need to use their own web-farm load balancers)
- Two machines for the Router server (one active, the other passive)
- Two or more machines for the Runtime server
- Two or more machines for the Task server (this is key if there are many publications and ETL processes)
- One machine for the repository database
- The data stack, to be analyzed, should be separate, including the In-memory database. These machines should be scaled out too.

Elastic Scaling

Pyramid may also be deployed via Kubernetes, offering dynamic scaling up and down, depending on the load applied to the Pyramid installation. For further information on Pyramid Kubernetes deployment, please consult the Kubernetes Deployment user guide.

The considerations discussed in this document regarding the different Pyramid Services and their resource requirements, also applies to a Kubernetes Deployment.

Key Server Designations

Real-Time vs Batch

The Runtime, Task and AI servers carry most of the server-side processing load. Getting the right balance between them can often be useful for resource allocation and server deployment decisions.

The most obvious guidance is to separate the two tiers on to at least two machines: runtime for real-time activities, task and AI for batch processing of data. As mentioned, real-time queries are the core of data analytics, so the Runtime engine is usually given the highest and strongest resources. Because time performance is less of an issue for running batch jobs, the Task and AI engines can be allocated less or weaker resources.

If there is high demand for batch processing of data and publications, another good strategy is to deploy at least 2 task servers. One should be designated for publications, the other for data preparation/ETL jobs (from inside the admin console). This will ensure that the heavier ETL jobs can be throttled separately to the lighter publication jobs.

Setting the task engine's peak/off-peak concurrent jobs will also provide the right balance between these two elements and the real-time querying requirements of users. Note that the runtime servers are used during data modeling (ETL) design sessions for generating previews. These processes can have a significant impact on the runtime server's performance for other tasks.

AI – Augmented Analytics and Machine Learning

If there is a strong reliance on AI and machine learning, it should also be given its own resources and machines. Both Python and R can be CPU and memory intensive depending on the size and complexity of the work. The NLQ component, which is also housed in the AI server is far less CPU intensive, but it can require a large memory footprint (2-6GB)

In-Memory

The bundled in-memory engine is designed to offer a simple alternative to other in-memory column-store database technologies like Microsoft Tabular and SAP Hana. Since it houses customer data, which is idiosyncratic in design, size and formulation, it is hard to provide specific guidance on servers hosting the in-memory engine.

However, as explained elsewhere, the in-memory server at least needs the requisite memory to host the analytic databases (see the Appendix for some guidelines). Beyond that, its performance is heavily affected by the number of available CPUs. As such, separating it from the other Pyramid tiers is highly recommended so it does not compete with other processing jobs.

Scaling Pulse

Pyramid Pulse is a standalone application server that is installed on a remote network, to allow Pyramid to connect and query data stacks hosted on the remote network. Its performance is designed to be very "light" – which means the engine itself does little work or processing. However, there are always limits.

When installing Pulse, customers can deploy a single instance on a single server which can then connect to multiple data sources on the network. This is the most efficient model when there are no performance constraints. On the other extreme, customers can elect to install multiple instances of Pulse on multiple remote servers. This then is the only scaling option for Pulse, since each data source can only use one Pulse node, and Pulse nodes cannot be clustered.

Performance Bottlenecks

Slow real-time querying

If your users feel that the real-time query responses are slower than expected, first ensure that the runtime server is not overwhelmed. The runtime server carries most of the weight. Usually, it is the first tier to be scaled up or out.

Follow that by a review of the data stack machine. Usually, if the runtime engine is not being taxed, performance is being degraded by the query response times in the data stack. Transaction logs can highlight this issue.

Lastly ensure that the batch jobs running on the task engine are not competing with users for resources. The easiest technique is to throttle the number of concurrent jobs the task engine can perform during user hours ("peak").

Overwhelmed Data Stack

If the data stack is overwhelmed, it may be under resourced. Apart from increasing the resources or scaling it out, ensure that the number of batch jobs submitted to the task engine are not overwhelming the engine – especially when users expect to interact with the system during peak hours. The easiest way to mitigate this issue is to use the task throttling settings in the admin console and selecting how many concurrent activities can run during peak and off-peak times.

Slow Content Management and Application

The entire application's performance can be affected by the health and performance of the repository database. Ensure the host server and database technology hosting the database is well resourced and that the database is fully tuned and operating on fast disks.

Other Factors

Another element is the speed and performance of the authentication engines such as Active Directory4. These elements slow all authentication processes, logins and database connections for queries. Related to this is the performance of networked services like DNS, which can slow server connectivity and communications.

Appendix
Load Testing Design

The following basic ideas are useful in designing load tests on the runtime engine. Testing of the task engine is less relevant since it is not designed for real-time responses and is throttled.

1. Login and content management: test the login in for users and use of the content tools. Do not include the login exercise in every test, since a user does not login each time, they want to do a specific activity.
2. Query Performance: test opening a discovery report together with several follow up ad-hoc interactions on the report. Repeat this for standard query sizes (100-500 cells); less for large queries (2000-10,000 cells); and seldom for excessive queries (100,000+ cells).
3. Calculation Performance: test reports that include formulae (custom members) and lists (custom sets).
4. Concurrency Performance: test dashboards that include several standard queries and slicers.
5. Mix testing: use a mixture of the above tests, since real world users

Database Repository Considerations

Many deployments underestimate the size and performance needs of the repository database and its host server. There is a need to ensure that this part of the cluster is extremely well resourced to prevent it becoming the bottleneck for the application.

All three supported technologies – Microsoft SQL Server, PostgreSQL and Oracle – are designed to support large, high performance data footprints and queries from thousands of concurrent requests. The strategies for scaling these out include a scale up model (bigger machine: more memory, multiple disks, and more CPU's) as well as a scale out model with clustering.

Deployment suggestions

- As a first step, Pyramid highly recommends installing the repository on its own server, so it does not compete with the other tiers for resources.
- If the usage is expected to be very large (high user concurrency), then a clustering approach should be considered.
- If Pyramid is deployed to the cloud, using a database service may represent the easiest and most effective technique for scaling out the repository database and its functions.
- Use very fast disks to house the database files. SSD's can have a significant impact on performance. Memory and CPU are less impactful.

Tactical Suggestions

- If all transactions are logged permanently and/or logs are stored for long periods of time, the log tables should be moved to their own filegroups on different physical disks.
- If verbose logging is not used, turning off benchmark, transaction and audit logs can reduce the weight of database activities not central to the product's operation.
- Ensure that settings related to connection concurrency are maximized. (PostgreSQL defaults are generally too low).
- Run back-up operations off hours.

In-Memory Size Guidance

An exact sizing technique for sizing databases hosted in the In-memory engine is complex. In, general, Pyramid's in-memory databases are like those produced by other column-store technologies like Microsoft SSAS Tabular and SAP Hana. The guides below are based on real-world examples and experience. But they could be quite different in each scenario.

- On disk the save database (in all these technologies) is compressed 7-10 times of the original relational data source size. The amount of compression on disk is driven by 2 factors:
    - **Data types** - integer's vs decimals vs dates vs text/string. Each data type has its own byte length and size. Integers are lowest through to strings which are heaviest. The heavier the data type, the more space it requires.
    - **Cardinality** - the number of unique values for a given column. If there is a high amount of repetition, the compression is higher. If each value in a column is entirely unique, there will be far less compression.
- In memory the database expands its footprint. A rough guide is that it will occupy **twice** its compressed disk footprint. So, the loaded database will be compressed 3-5 times of its original data source size in memory.

Using many sample databases, Pyramid has found that a rough rule of thumb is that a 10 million row fact table may take 2Gb of memory.

In-memory performance is affected by these resources:

- **CPU**: CPUs are used to crank query results, and because there is no "read" time like on-disk databases, the CPUs are the main limiting factor in processing requests. The number of concurrent requests that can be handled, therefore, is a simple function of the number of cores at the engine's disposal.
- **Memory Quality**: Although this far less impact than the speed and number of CPUs, faster memory will obviously load and process tasks quicker than older memory modules.
- **Disk Speed**: this has a very minor role for housing the saved databases. Load up speed and save speed however are affected by disk speed. For very large systems, this can impact the build and load up time considerably.

Pyramid's Baseline Load Testing Results

The following results are based on Pyramid's internal baseline load tests.

Single Machine Tests:

- Hardware: 8x i7 CPU, 16Gb DDR3 Ram, 512Gb SSD Drive
- Software: The entire application is installed on a single machine, including PostgreSQL for the repository and the In-memory engine

Dual Machine Tests:

- Hardware: Dual 8x i7 CPU, 16Gb DDR3 Ram, 512Gb SSD Drive
- Software: The entire application is installed on a single machine, including PostgreSQL for the repository and the In-memory engine. Only the Runtime engine is installed on the second machine.

Common:

- External Factors: Separate Active Directory servers; Gigabit Corporate LAN
- Load Testing Software: Visual Studio 2015 Desktop
- Host OS: Windows 2016 Standard
- Basic Authentication used for websites with Active Directory