happiest minds

A Comparative Study on Vega-HTTP & Popular Open-source Web-servers

Contents

Abstract

In this paper we present our lightweight, high-performance, low-latency web-server called VEGA-HTTP, its architecture and compare its performance with other existing popular open-source web-servers. Along with the existing platform we are also sharing things we have planned to develop in the future.

Introduction

A web-server is a computer program that is responsible for accepting HTTP (Hypertext Transfer Protocol) requests from the clients and serving them with HTTP responses along with optional contents which form web pages to the users, such as HTML documents and linked objects (images, etc.).

Our Requirements: Handle high concurrent connections with extremely low latency and very less memory usage.
Our Challenge: The existing servers failed to the above requirements at a very high load without utilizing the system resources to the fullest.
Our Solution: The VEGA-HTTP server was built with a goal to overcome these challenges. This server has been built using the Linux kernel call epoll with the edge triggered mode at its heart. This enables it to handle the requests and maximizes performance by giving complete control of all the kernel level tuning parameters for better optimization of the server.

Performance Comparison

The Performance Comparison in our test environment has been very encouraging. The results of the comparison showed how a modicum server can perform better than most of the open-source servers, available on the Internet.

System Configuration:	
• CPU :	intel i3 3.3Ghz (2 cores)
• Processors :	4
• RAM :	8GB
• OS :	Xubuntu 14.04 x86_64
• Test Suit :	WRK

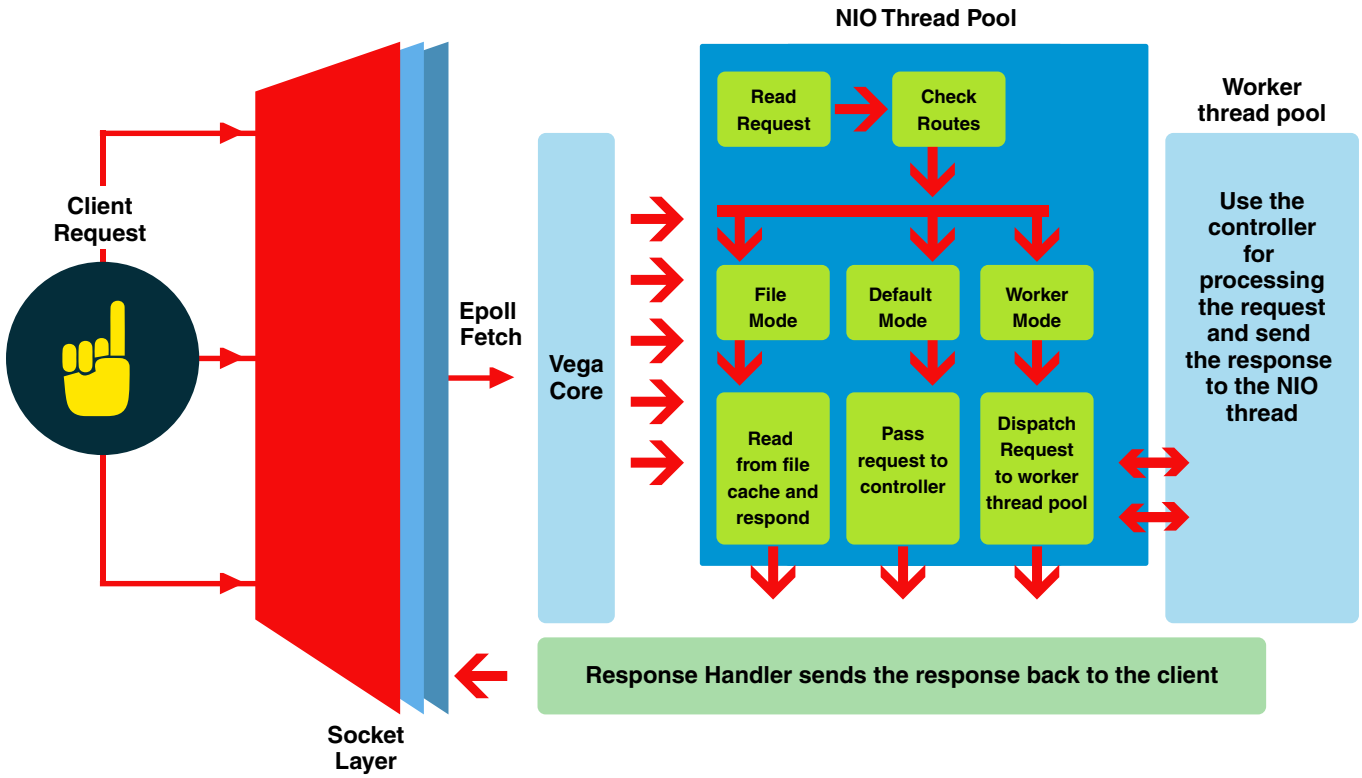
We have three different types of tests with concurrencies 100 and 1000 for a period of 10 minutes.

- File Serving
- A Default mode or NIO mode Test
- A Worker Mode Test

Benchmark Results:

Http-Server	File Serving 100c	File Serving 1000c	Hello World worker Mode 100c	Hello World worker Mode 100c	Hello World Default Mode 100c	Hello World Default Mode 1000c
Vega-http	192129	160389	106568	114070	201620	173791
NxWeb	173742	140142	88297	86894	178489	148050
Netty	NA	NA	NA	NA	139940	131685
Undertow	38297	38198	105883	106257	158920	137508
Rest Express	NA	NA	NA	NA	80093	78940
Nginx	64854	60383	NA	NA	NA	NA
Apache2	33410	32730	NA	NA	NA	NA
Tomcat Servlet	NA	NA	NA	NA	31826	29953
Tomcat JSP	NA	NA	NA	NA	12499	10302

Architecture Diagram



Vega Core

- Using Linux kernel's epoll, Vega Core loads incoming client requests in a batch manner and dispatches request (based on configuration) to network I/O threads (NIO threads) for processing
- Linux system call epoll has support to wait on a particular set of open socket/file connections and once the request is ready epoll dispatches it to the NIO threads for processing and returns the response.
- The Kernel internally handles synchronization issues and only dispatches the complete requests that it has received.

NIO Thread Pool

The network input/output thread pool (NIO thread pool) has three tasks in the Vega server:

- **Reading the request:** Network thread fetches incoming request from open socket connection and parses the HTTP-headers and its HTTP-body content from the request
- **Processing the request:** The parsed headers request path is matched against the preloaded routes cache. There are three routes caches one for each mode (Default /Worker /File). Each path is configured with mode, path controller func tion or filename to serve in the route configuration file. Based on the configured mode they are loaded into either of the following cache's Default (or NIO) Mode, Worker Mode, and File Serving mode.

Some Key Decisions while Designing:

For Processing, the routes cache is based on 'The Trie Data Structure' where the key is the route path and value are the index points to the respective function pointers in the function pointer cache. Function pointer cache is normal in memory array which gets populated on server startup with all controller functions for faster execution.

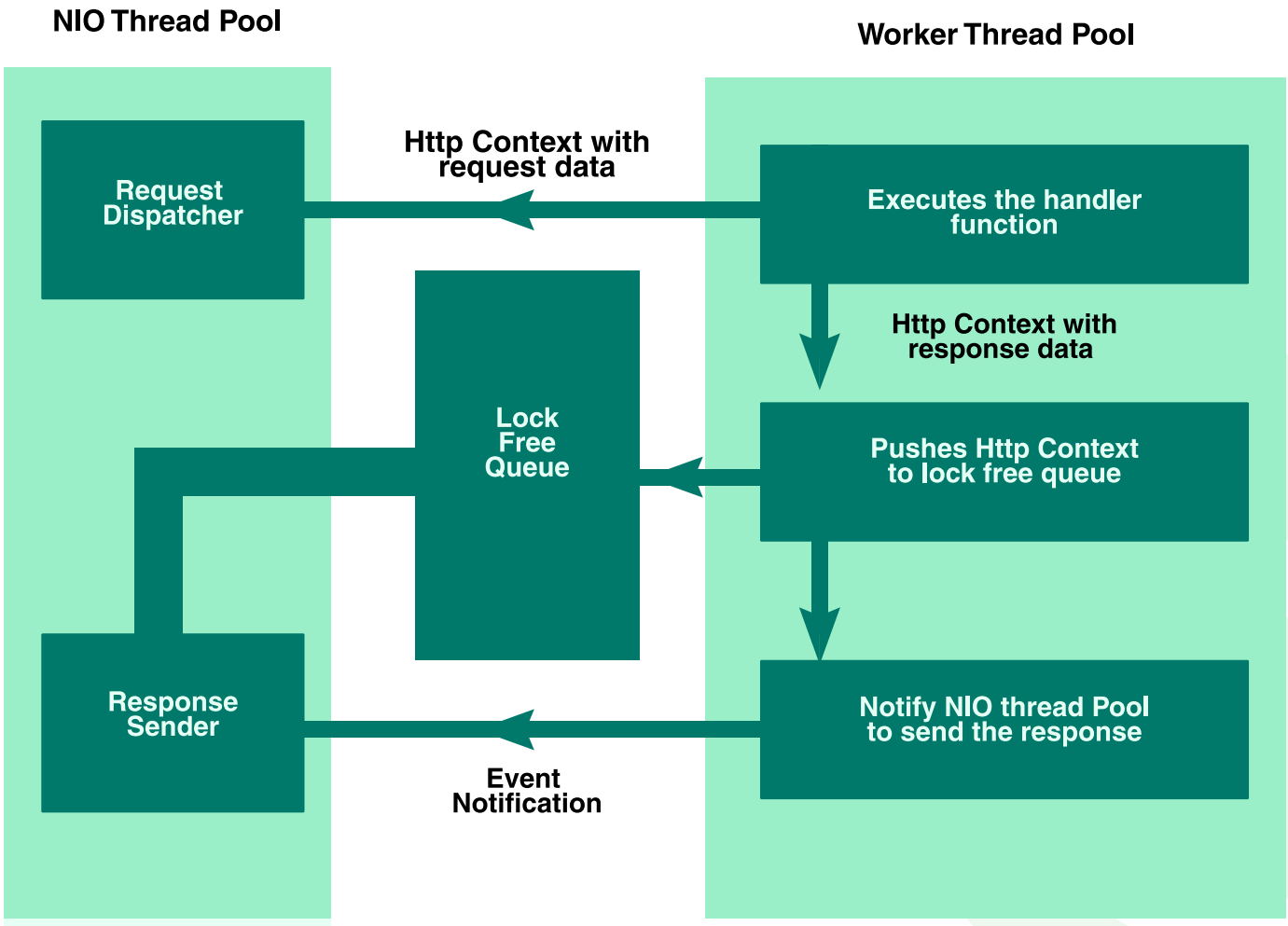
For file serving instead of the function pointer, it contains the file contents already loaded in the memory for quick serving. The File contents cache is populated on server startup with all file contents uploaded in memory.

Other than the file cache all the other routes and functions cache are loaded per NIO thread so that they do not have to share any resources even with any other thread not even in read-only mode.

Default Mode or In-Thread mode

In the request life cycle after the route has been matched and if the route has been configured for default (NIO) mode, then the controller function is loaded in the same NIO thread which received and parsed the http request to process the request and the response context from the controller is sent back to the client as a http response.

Worker Mode



In the request life cycle after the route has been matched and if the route has been configured for worker mode: The Network thread dispatches the request to the worker thread pool for processing the request. When the request is dispatched the NIO thread also passes the request object to the controller. The worker thread then executes controller function and stores response body in the context passed to it. After execution of the function, the worker thread has to send context data back to the network thread to respond back to the incoming request. For this purpose, an intermediate lock-free queue is used. The worker thread pushes the context data to the intermediate queue. The lock-free queue is common for all network threads and worker threads. To prevent thread locks and to reduce the latency lock free queues are used. After pushing context data to the queue, worker thread sends a notification to network thread via epoll with eventfd. Eventfd is a file mechanism which usually used for faster inter-thread messaging and it is supported by epoll. All network threads always wait for open socket/file connections from epoll. If the epoll triggers a network thread with eventfd file then that particular network thread reads data from the queue and responds back with the respective data to the incoming request.

File Serving Mode

All file contents are loaded in file-serving cache on server startup itself. When network thread recognizes the requested with file serving mode it directly loads the content from the cache and sends proper response headers to the client.

Architecture Choices

'C' over 'C++' and 'Java'

Linux system calls can be used flexibly in 'C' over 'C++'. Cross-language over system calls causes unnecessary buffer copying which will affect the overall performance of the server. This application requires more buffer to buffer transfers. JVM specific languages don't support some high performing system calls for file monitoring. Garbage collection affect application performance which completely forces JVM languages for out of system application programming. The JVM also doesn't have support for Epoll edge trigger mode.

Epoll over Select and Poll

In Linux environment, each socket connection is a file but doesn't support random access. There is a requirement in servers to notify when there is data available in open files with its file descriptors. To achieve this three familiar system calls are poll [3], select [2], and epoll [4]. Poll [3] performs faster when there is less number of files to monitor. Select [2] performs faster even when there is more number of connections but there is a requirement to fix the maximum number of files to monitor. Select [2] and Poll [3] doesn't support edge triggered mechanism. These two limitations are overcome in epoll command. Epoll is fast, flexible and supports various mode of operations.

Epoll edge triggered mode over Epoll level Triggered mode

Epoll supports two modes of operations Edge trigger and Level trigger:

In Edge trigger mode when there is data available on any of the socket/file connection it will trigger hook function once. And all network threads listen to that hook function. In level trigger mode hook function gets called until data gets flushed. So if several threads listen to epoll all threads epoll hook function will get triggered. This causes synchronization and performance problem. The synchronization problem can be at least handled by preventing the mixing up of other threads with the first thread which actually fetches data. However performance problem will occur in level trigger mode because if all thread wakes up from thread pool all those require CPU time and unnecessary context switching will take place in processor level to provide CPU time for all threads. If Level trigger mode is used, a request can be handled by one thread but all other network thread will also be awake. If edge trigger mode is used a request can be handled by one thread and at the same time only that thread alone will be awake. Several High performance C servers use libev, libevent as its event managing multiplexing library. We stick with plain system call usages to prevent unnecessary abstraction which showed its part in performance.

Trie over Hash table

The internal caches in our server completely uses Tries and plain c arrays. We have tested with glib hash table and glib tries for our cache. The performance of the glib tries was much greater than the hash table and hence we decided to use trie map.

To improve the performance we are using a tire map which would have an index stored as the response of a trie lookup. The given index is used to access the data Object which is stored in a fixed size array. The change from a traditional hashMap implementation to a trieMap for routing the increased performance of route lookups by 10% which is a significant number for such a small change in data Structure.

The trie map is an ordered tree data structure that is used to store a dynamic set or associative array where the keys are usually strings. Unlike a binary search tree, no node in the tree stores the key associated with that node; instead, its position in the tree defines the key with which it is associated. All the descendants of a node have a common prefix of the string associated with that node, and the root is associated with the empty string. Values are not necessarily associated with every node. Rather, values tend only to be associated with leaves, and with some inner nodes that correspond to keys of interest.

Initially when the server is being started it loads the routes into three different trie Maps one for each mode W - for the Worker mode of execution N - for the Network Thread mode of execution and F - for the file handler mode of execution. All the files are loaded into memory for quicker delivery. The route path and the files in the given directory is stored in the trie map for delivery. The index of the trie lookup is is pointed to the location where the files are stored in memory.

Centralized queue and eventfd over Pipes for message passing

Pipes is heavy when compared to eventfd. Eventfd doesn't support the direct transmission of data which makes it fast over pipes. Pipes require heavy system calls to copy data to pipes. Instead of pipes, adding data to an in-memory location is faster, hence, we preferred queue. Also, pipes forces to use heavy system calls in worker threads which will force context switching and reduce application performance. We used lock-free queue which is faster when used in between multiple threads which prevent thread locks.

Optimization Options

Thread Pool Size

There are two different pools which can be optimized depending on the tasks that the server has to do.

Network Thread Pool which is also referred to as NIO Thread Pool is the number of threads that receives and responds to the client request. The Default mode of execution happens within these threads. The Optimized choice for the thread pool size for maximized performance is the number of processing units that are available in the system. This can be found by running the bash command nproc in a Linux environment.

Worker Thread Pool size is the number of threads that are allocated for the worker thread to execute the controller function. This can be set to higher level depending on the tasks that the controller has to perform. It is advised to perform IO operations in the worker thread mode to avoid blocking calls from being executed within the NIO thread pool as this will affect the number of requests that the server can handle directly. It is also advised to perform expensive operations in the worker thread mode to increase the throughput of the server.

Message passing File Handlers size

The message passing file handlers are used to communicate between the worker thread and the NIO thread to pass the http_response_context. If the number of requests that are being dispatched to the worker thread pool is high it is recommended to use a higher number of file handlers. The optimized number of file handlers is half of the network thread pool size. Note that this cannot be higher than the number of NIO threads.

Epoll Initial Batch Size

The Epoll Initial Batch size is the maximum number of connection descriptors you expect to manage at one time. The kernel used this information as a hint for the amount of space to initially allocate in internal data structures describing events. (If necessary, the kernel would allocate more space if the caller's usage exceeded the hint given in size.) This size should be greater than zero.

Epoll Time Wait

The epoll time wait parameter is one of the two flags where the epoll_wait [1] command will exit returning all the events (requests) that are ready to process. The epoll time wait specifies the amount of time that the epoll has to wait before flushing the ready events (requests) for processing. The two variables that should be considered for setting this value are size of the operations in the handlers and the load the server receives.

If the load of the server is high and the operation time is less it is recommended to have a higher time wait. If the load is less than it is recommended to have a smaller time wait. Note it is recommended to modify this parameter while load testing your handlers for different concurrency levels for better optimization.

Epoll Max Events to Stop Waiting

The epoll max events to stop waiting is the second of the two flags where the epoll_wait [1] command exit returning the number of events (requests) that is specified in the max events to stop waiting option. The two variables that should be considered for setting this value are size of the operations in the handlers and the load the server receives.

If the load of the server is high and the operation time is less it is recommended to have a higher number of max events. If the load is less than it is recommended to have a smaller number of max events. Note it is recommended to modify this parameter while load testing your handlers for different concurrency levels for better optimization.

Future Enhancements

- **Language Support**
- **PHP**
- **Ruby**
- **Python**
- **SSL Support**
- **Multi Domain Support**
- **Proxy_pass (Load Balancer)**
- **File Streaming (For serving media and large files)**
- **Connectivity with existing frameworks**
- **WSGI**
- **FastCGI**

References

- **epoll_wait – (http://linux.die.net/man/2/epoll_wait)**
- **poll – (http://linux.die.net/man/2/poll)**
- **select – (http://linux.die.net/man/2/select)**
- **epoll – (http://linux.die.net/man/4/epoll)**

About the Author

Suresh Raja

Data Analyst with Happiest Minds. His key interest is in building high performance computing environments which involves faster request processing, data streaming and analytics systems. Have worked for Digital Advertising industry where main challenges are running complex algorithms on huge data and that too on real time.

Vishnu Prasad

An expert in both Web based/Backend distributed, n-tier and Client/Server Applications. Having sound domain knowledge in Digital Advertising Industry and large scale Big Data processing, Analytical and Machine Learning systems. Highly passionate in building large scale distributed systems which could handle huge volumes of data.

Happiest Minds

Happiest Minds enables Digital Transformation for Enterprises and Technology providers by delivering seamless Customer Experience, Business Efficiency and Actionable Insights through an integrated set of Disruptive Technologies: Big Data Analytics, Internet of Things, Mobility, Cloud, Security, Unified Communications, etc. Happiest Minds offers domain centric solutions applying skills, IPs and functional expertise in IT Services, Product Engineering, Infrastructure Management and Security. These services have applicability across industry sectors such as Retail, Consumer Packaged Goods, Ecommerce, Banking, Insurance, Hi-tech, Engineering R&D, Manufacturing, Automotive and Travel/Transportation/ Hospitality. Headquartered in Bangalore, India, Happiest Minds has operations in the US, UK, Singapore, Australia and has secured $52.5 million Series-A funding. Its investors are JPMorgan Private Equity Group, Intel Capital and Ashok Soota.

E-mail: Business@happiestminds.com
Visit us: www.happiestminds.com

Follow us on