Lyra Metrics – Synthetic Load and Performance Tool

Lyra Metrics is an Akamai homegrown application that performs load testing and performance measurements on multiple resources (URLs) specified in a configuration file (config.json). The application measures various metrics that contribute to an entire http request for a given testObject, such as DNS lookup time, TCP connection time, TLS handshake time, server processing time, content transfer time and total time.

The results are either displayed in the console or written to InfluxDB a time series database deployable locally or in the cloud, depending on the value of the "use_influx_db" configuration parameter. The application uses multiple goroutines to simulate a configurable number of virtual users and launches these goroutines simultaneously. Overall, the application is an organized attempt to provide a holistic synthetic view of granular client performance metrics.

1. **DNS Lookup**: This metric measures the time it takes to resolve the hostname of the resource's URL to an IP address.
   a. DNS Lookup: This metric can be compared to the `nslookup` or dig command, which are used to perform DNS lookups.
   b. DNS Lookup: This metric is calculated as the difference between the time when the DNS lookup started and the time when the lookup completed. The calculation is performed using the **time.Since** function, which returns the time elapsed since a given start time:

```
dnsStart := time.Now()
_, err = net.LookupHost(req.URL.Hostname())
if err != nil {
return result{Error: err}
}
```

- `dnsLookup := time.Since(dnsStart)`

2. **TCP Connection**: This metric measures the time it takes to establish a TCP connection to the resource's server.
   a. TCP Connection: This metric can be compared to the telnet or `nc` command, which are used to establish TCP connections to servers.
   b. This metric is calculated as the difference between the time when the TCP connection started and the time when the connection completed. The calculation is performed using the **time.Since** function:

```
tcpStart := time.Now()
conn, err := net.Dial("tcp", req.URL.Host)
if err != nil {
		return result{Error: err}
}
defer conn.Close()
tcpConnection := time.Since(tcpStart)
```

3. **TLS Handshake**: This metric measures the time it takes to negotiate a secure connection using the TLS protocol.
   a. TLS Handshake: This metric can be compared to the `openssl` command, which is used to perform SSL/TLS handshakes.
   b. TLS Handshake: This metric is calculated as the difference between the time when the TLS handshake started and the time when the handshake completed. The calculation is performed using the **time.Since** function:

```
tlsStart := time.Now()
tlsConn := tls.Client(conn, &tls.Config{
		InsecureSkipVerify: true,
		ServerName:	req.URL.Hostname(),
})
defer tlsConn.Close()
err = tlsConn.Handshake()
if err != nil {
		return result{Error: err}
}
tlsHandshake := time.Since(tlsStart)
```

14. **Server Processing**: This metric measures the time it takes for the server to process the HTTP request and generate a response.
    a. Server Processing: This metric can be compared to the curl or `wget` command, which are used to make HTTP requests to servers.
    b. Server Processing: This metric is calculated as the difference between the time when the server started processing the request and the time when the response was received. The calculation is performed using the **time.Since** function:

15. **Content Transfer**: This metric measures the time it takes to transfer the response body from the server to the client.
    a. Content Transfer: This metric can be compared to the `pv` or `pipebench` command, which are used to measure the speed of data transfer.
    b. Content Transfer: This metric is calculated as the difference between the time when the content transfer started and the time when the transfer completed. The calculation is performed using the time.Since function:

```
contentStart := time.Now()
res, err := client.Do(req)
if err != nil {
		return result{Error: err}
}
defer res.Body.Close()
_, err = io.Copy(io.Discard, res.Body)
if err != nil {
		return result{Error: err}
}
contentTransfer := time.Since(contentStart)
```

26. **Total**: This metric measures the total time it takes to complete all the above steps for a single resource.
    a. Total: This metric is a combination of all the above metrics, and can be compared to the time command, which is used to measure the execution time of a command.
    b. Total: This metric is calculated as the difference between the time when the resource measurement started and the time when the measurement completed. The calculation is performed using the time.Since function:

```
start := time.Now()
...
return result{
		...
		Total: float64(time.Since(start).Milliseconds()),
		...
}
```

33. **Availability**: This metric represents the HTTP status code of the response, indicating the availability of the resource. For example, a status code of 200 means the resource is available and a status code of 404 means the resource was not found.
    a. Availability: This metric can be compared to the curl or `wget` command, which can be used to check the HTTP status code of a resource.