Node-DC-EIS Design document

Thursday, April 06, 2017 8:55 AM

Architecture Diagram:
https://sharepoint.amr.ith.intel.com/sites/DSLO/_layouts/PowerPoint.aspx?PowerPointView=ReadingView&PresentationId=/sites/DSLO/Shared%20Documents/Workloads%20Leadership%20DSLO%20chapter/Node-DC-Architecture-diagram.pptx&Source=https%3A%2F%2Fsharepoint%2Eamr%2Eith%2Eintel%2Ecom%2Fsites%2FDSLO%2FSitePages%2FHome%2Easpx%3FRootFolder%3D%252Fsites%252FDSLO%252FShared%2520Documents%252FWorkloads%2520Leadership%2520DSLO%2520chapter%26FolderCTID%3D0x012000BD68597E2F6BB54E9D140508C6E7F079%26View%3D%7BD7636946%2D3910%2D4C5A%2DBD09%2D74BF1B18EDA5%7D&DefaultItemOpen=1&DefaultItemOpen=1

Scope and Goals

- Simulate a relevant application model.
- Exercise the components of the Node.js runtime and it's OS and hardware environment
- Include a "requests per second" and "response time"

Scope and Goals

- Scaling
  - In this benchmark, master process interacts with worker to schedule a task (user request), this allows the workload to scale across multiple-cores.

- IO Components
  - Node-DC benchmark exercises the CPU, memory, network IO and disk I/O.

- Run Time
  The average run time is 2 minute which is a configurable parameter. This benchmark has multiple phases:
  - Setup
  - Database creation
  - Ramp-up
  - Measuring window (time)
  - Ramp-down
  - Data collection/Report

Workload Architecture

Server Modeling
The application scenario chosen for the Node-DC workload is a world-wide typical Employee Information System.

- The system stores Employee personal information, such as,
  - Name
  - Address
  - Family
  - Compensation
  - Health
  - Photo

- Queries employee information via,
    - Unique primary key (id),
    - Last name, and
    - Zipcode
- Posts new employee record
- Delete an employee record by unique primary key (id)

The workload can be deployed in a multiple ways. See deployment section for more details.

Server Components

- Models
    - Application in default case mimics standard relational database schema design with multiple models (employee, address, family, compensation, health and photo), connected with employee_id as a foreign key. Database used is 'mongodb'.
    - In additional use case scenarios it uses one model with nested object support from NoSQL database (mongodb).

- Controller
    - The controller directs the execution of the callback. There are two controllers in single instance and multi-process mode. One to populate the database with seed data and second one containing all the action callbacks. Controller contains business logic code that processes client (service) requests , queries data store and responds to client.

- Routes
    - Routes handler directs each request to appropriate controller action.

- Data store
    - For data store 'Mongodb', a NoSQL database has been used, which is a default choice for Node.js based application.

Use Cases and deployment strategies

Use Cases

- JSON data exchange
- Server Side Rendering (SSR) with,
    - Default PUG (aka Jade) template engine
    - React.js
    - Electrode framework

Server Deployment

- Single Instance
    - One Node.js runtime instance
    - Separate Mongodb server process either on the same machine(node) or different node
    - If process dies for some reason complete runtime dies

- Multi-Process Instance (Cluster mode)
    - This mode is useful to take advantage of multiple cores if available in the test system.
    - Comprise of multiple Node.js runtime processes. Contains one master process and remaining slave/worker processes. They communicate using 'shared memory' IPC mechanism.
    - All processes listens on the same port.
    - If one process goes down, it can continue to function as long as there are worker processes are available.

        - In peak load of traffic, Node will automatically allocate the worker to particular process (load balancing effect)

    - Multi-Service Instance (Micro-Services mode)
        - Application is divided by functional boundary refactoring into multiple Node.js services or individual Node.js application. Each service has its own routing, controller and backed logic. They are independent process but together provided same functionality as the single instance. These services can be deployed on the same node or across multiple nodes.

    - Multi-ServerApp-instance (multiple single or cluster mode)
        - This mode is used to saturate system under tests with maximum CPU utilization and test application scaling performance.

Workload Setup and Execution (Requests and Transactions)

- Setup
    -
- Workload Execution Phases
    - Ramp-up phase:
    - Measurement phase:
    - Ramp-down phase

Metrics Computation

Client Driver Details

runspec.py

- Command line options support for,
    - Number of requests,
    - Number of concurrency
    - Amount of time for time based measurement
    - Optional - Top level logdir
    - Optional - Instance id
    - Optional - Debug flag
    - Optional - A Configuration file in JSON format

- Contents of config file,
    - client_params:
        - #of requests
        - #of ramp_up_ramp_down_requests
        - #of concurrency
        - #time for time based measurement
        - Temporary log file
    - server_params:
        - Server ip address
        - Server port number
        - Server root url
    - db_params:
        - #of total records to populate
        - Ratio of distribution of 'name' attribute for Node-DC
        - Ratio of distribution of 'zipcode' attribute for Node-DC
    - url_params (total must be 100%):

        - Total URLS/endpoints to exercise
        - Ratio of 'name based urls'
        - Ratio of 'zipcode_based urls'
        - Ratio of 'id_based_urls'
        - Ratio of 'GET' requests
        - Ratio of 'POST' requests
        - Ratio of 'DELETE' requests
    - Memory_data_collection_params:
        - "memstat" interval in seconds
        - "memstat" log file
- Runspec phases
    - Setup
        - Either parsing a config file, or
        - Parsing command line arguments
        - Creating log directory structure
        - Capture CPU or HW information
    - Import 'Node-DC' specific module (exposing server API - "main_entry()")
    - Populate database (monitor and wait till it's successful)
    - Build url list (and shuffle)
        - Build list of ids, names, zipcodes
    - Create Pool of threads (matches given concurrency)
    - Start a process to capture "server memory statistics"
    - Start a process to process temporary log files generated during the run
    - Start Run
        - Ramp up phase
            - Make server requests either fixed number or for time
        - Measuring Phase (steady state of an application)
            - Make server requests either fixed number or for time
        - Ramp down phase
            - Make server requests either fixed number or for time
        - Common functionality in each phase
            - Record start_time and "phase" name before the request
            - Send the request
            - Record end_time
            - Record request details.

    - Stop the run
    - Validate the run
        - Check database
    - Stop the Server
    - Post process log(s)
        - Generate memory statistics
        - Generate throughput, latency and concurrency metric - summary text file
        - Generate graphs highlighting the behavior

- There is additional setup, processing requires in "Multi-ServerApp-instance" mode of execution.
    - There is master orchestrator program/script called "multi_instance_run.sh"
      This script works with its own (special) configuration which has multiple JSON data blocks with information about each instance of server application. There will one block per instance user wants to run.
        - A server block contains,
            - Server IP address
            - Server Port number

        - Optional 'data base server ip address'
        - Optional 'data base server port number'
        - Optional 'data base name'
- Master script parses does following,
- Clones 'Node-DC' client code from github.com
- Clones 'Node-DC' server code from github.com
- Parses main configuration file.
- Generates new client config_<instance_id>.json config file per instance block
- Generates new server config file in the server/config directory with appropriate changes
- Remote copy modified server code to the specified server (requires autologin setup for ssh)
- Repeat for each server instance block in the master config file
- 'Remote start' each server instance
    - Start database server instance if require
    - Start Node-DC application server
    - Check for any error
- Exit the run and notify user if any error occurs
  ***At this point we expect that all instances are running successfully**.
- Execute runspec.py with config_<instance_id>.json input file. Repeat for all client configurations.
- Co-ordination details:
    -