Neural Network Technology for Real-Time IT Service Management

Viktor Krasnoproshin
Faculty of Applied Mathematics and Computer Science
Belarussian State University
Minsk, Belarus
Email: krasnoproshin@bsu.by

Aleksandr Starovoitov
Faculty of Applied Mathematics and Computer Science
Belarussian State University
Minsk, Belarus
Email: StarovoytovAA@bsu.by

***Abstract*—The paper explores a relevant applied problem related to building decision-making systems for resource management of critical IT services. The uncertainty of external load is an important factor that affects operational management. Neural network forecasting is used to improve control systems. A model system of a critical IT service is described, an original technology, structure and architec- ture of the control system are proposed. Experiments have been conducted to confirm the workability of the proposed technology.**

***Keywords*—decision-making, information system, proac- tive management, uncertainty of external load, neural networks, critical IT service**

I. Introduction

Support for computing systems in critical infrastructures (such as banking, telecommunications, industrial systems) in an operational state (with guaranteed computational resource levels) is a relevant problem in today's digital society.

Uncertainty in external loads and outages of computing equipment lead to operational failures and performance degradations of critical IT systems. As a result, the loss of operational efficiency in processing information and conducting banking and other operations can have serious consequences, including financial losses and major incidents.

Making operational decisions for the management of critical IT services allows for the reduction or prevention of negative consequences. However, the human factor often contributes to a decrease in operational efficiency. Therefore, various automated solutions are actively being developed to enhance efficiency and proactivity in the management of critical systems.

In laboratory conditions, it is difficult to develop relevant systems (without interacting with the critical infrastructure itself). Therefore, one of the current problems is the creation of model systems that enable researchers to use them for the development of proactive management systems for critical IT services.

Several authors develop various management systems for critical IT services using neural network models, which contribute to efficient decision-making [1]–[4]. However, in these systems, only one neural network model is trained for specific types of external load. It is assumed that these models will successfully forecast the values of necessary parameters for other types of load associated with uncertainty. In these works, training datasets are prepared in advance, containing long time series with a large number of elements. Training on such datasets takes a considerable amount of time and requires high-performance resources (GPU, TPU) to effectively train models within an acceptable timeframe.

This paper considers the construction of a model system conceptually corresponding to a critical IT service and an operational management system with a neural module.

A multi-model approach is used based on the idea that the managed IT system can be in various states (in terms of computational resource volume), and for each state during its existence, a neural network model can be created and trained to predict the average %CPU utilization across computational modules.

A combined management system is used, in which the control decision for state changes is formed based on both reactive and proactive approaches.

II. Critical IT Service: Conceptual Model

Let's consider the conceptual model of a critical information system, which describes its basic elements and significant parameters. It can be described in the form of the following tuple:

$$\text{System} = (\text{Clients} \cup \text{Balancers} \cup \text{APPs} \cup \text{DBs}, \text{Links}), \quad (1)$$

where

Clients = $\{\text{Client}_i\}$ — the set of system clients. These can be both user devices and other external systems;

Balancers = $\{\text{Balancers}_j\}$ — the set of balancers that distribute requests from clients and external systems across services. Services within the system can also communicate with each other through balancers;

APPs = {APP_k} – the set of application services within the system;

DBs = {DB_l} — the databases of the system (can be of any type);

Links = {$Link_m$} — the set of bidirectional temporal links that arise during communication between elements of the system;

The elements $Client_i$, $Balancers_j$, APP_k, DB_l, $Link_m$ can also be represented as tuples:

$Client_i$ = ($Protocol_i$, $Profile_i$), where $Protocol_i$ defines the specification, and $Profile_i$ represents the interaction profile (requests, their parameters, frequency, etc.) for the i^{th} client;

$Balancers_j$ = ($Protocol_j$, RPS_j , $Throughput_j$), where RPS_j is the number of requests per second, and $Throughput_j$ is the throughput (Mbps) for the j^{th} balancer. It is assumed that the balancer supports all interaction protocols, and the delay on the balancer is insignificant compared to the response time;

APP_k = ($Compute\_Modules_k$, $Soft\_Service_k$) the set of instances of application software, where $Compute\_Modules_k$ is the set of computational modules, and $Soft\_Service_k$ is the type of application service used on the computational modules $Compute\_Modules_k$;

DB_l = ($Compute\_Modules_l$, $Soft\_Service\_DB_l$) — describes a set of database instances serving one type of application service instances, where $Compute\_Modules_l$ is the set of computational modules, and $Soft\_Service\_DB_l$ is the type of database software used in the computational modules $Compute\_Modules_l$. These can be any (not necessarily relational) databases;

$Link_m$ = (P_{mn}, P_{mo}) — the set of bidirectional temporal links between elements of the system, established through communication via specific open ports;

Compute_Modules = {$Compute\_Module_c$ } — the set of computational modules of the system, where each computational module can also be described by the following tuple:

Compute_Module = (CM_Type,
CM_CPU_Limit, CM_CPU_Perf,
CM_RAM_Limit, CM_RAM_Perf,
CM_Storages, CM_NET_Throughput),
where

- CM_Type ∈ {"physical server "logical or hw partition "virtual machine "container"} — the type of module;
- CM_CPU_Limit — the number of processors (CPU or vCPU) available to the module;
- CM_CPU_Perf — the maximum performance of one processor in the module;
- CM_RAM_Limit — the available volume of RAM in the module (GiB);
- CM_RAM_Perf — the maximum performance of RAM in the module (determined by bandwidth and memory access time);
- CM_NET_Throughput — the maximum throughput capacity of the module (Gbps).

CM_Storages = {$CM\_Storage_p$ } — storage modules available to the computational module. Each storage module is defined by a tuple:

CM_Storage = (CM_Storage_Type,
CM_Storage_Capacity, CM_Storage_Perf), where

- CM_Storage_Type ∈ {"local "external"} — determines the type of connection of the storage module to the computational module. In this case, the local option describes local disk resources (HDD, SSD). The "external" component refers to storage resources external to the computational module. These can be connected using various input-output devices that support different block protocols (iSCSI - SCSI over Ethernet, Fiber Channel Protocol – SCSI over Fiber Channel, NVMe-oF – NVMe over Fiber Channel), file protocols (CIFS, NFS) and object protocol (S3).
- CM_Storage_Capacity — the storage capacity of the module (GiB).
- CM_Storage_Perf — the performance of the module can be defined as the number of Input/Output Operations Per Second (IOPS) (with minimum response time) or throughput (GBps), for different workload profiles.

The workload profile is determined by the percentage of read operations (%Read), the size of data blocks (KiB), and the distribution, which indicates the presence of block sizes in requests, their proportion, and the delays associated with them in the total stream of requests.

Soft_Service = {Soft_Servicek} — types of application services in the system used by computational modules in the system.

Soft_Service_DB = {Soft_Service_DBl} — types of database software used in computational modules.

In information systems, for integrating various services, elements such as message brokers (e. g., RabbitMQ, IBM WebSphere MQ, Artemis, Apache Kafka, etc.) are often used. In our case, these services are not allocated to a separate class since essentially they can be attributed either to the set of entities in APPs or to DBs if this functionality is implemented at the database level (e. g., Oracle Advanced Queuing).

The model structure explicitly does not include: telecommunication equipment (Switches, Routers), resource management systems, and various perimeter control system of the information system. All of these may restrict access to the system from outside and between components based on ports and protocols (Firewalls). Additionally, they may lead to deeper inspection of the exchange at the application level (WAF — Web Application Firewalls) and the necessity to perform analysis of exchanges at OSI Model layers 3 and 4 (IPS — Intrusion Prevention System), tracking anomalies and conducting

checks for known vulnerabilities and attack vectors based on signature databases and established policies.

The structure also explicitly does not include monitoring and logging systems, antivirus protection, encryption, as well as other technological services and systems (LDAP DNS, backup and recovery services, CI/CD systems, time synchronization service, etc.). It is assumed that these services are properly configured, and their operation does not affect the functionality of the system being considered.

III. Model System: Task Statement

With consideration of the described model, the following task is formulated:

To develop the system that conceptually corresponds to the previously described model of a critical information system (1), meeting the following criteria:

- The system can operate on a workstation, laptop, or virtual machine with resources not exceeding 4 CPU cores and 8 GiB of RAM;
- The set of APPs is represented by a compact web application capable of handling external requests;
- An instance of the web application service operates within the computational module Compute_Module, which has a specific limit on CPU resources (other limits are also possible but not mandatory);
- The system has the capability to scale within the specified resources mentioned above, meaning the number of instances of the web application service can be adjusted during operation under load by increasing or decreasing the number of computational modules. Scaling management is available both programmatically and manually;
- For each computational module of the system CPU utilization metrics are collected, with a metric collection period $\sim 2s$. The data is stored in CSV files with the specified frequency. When additional computational modules are added, statistics collection is automatically enabled for them;
- The system should have a reactive scaling module that operates according to the following algorithm: after receiving metrics of %CPU utilization for the running computational modules hosting an application web service, the average utilization percentage $\overline{\%\mathrm{CPU}}_N$ is calculated based on the number (N) of running modules. If $\overline{\%\mathrm{CPU}}_N > 50\%$, the system automatically adds another computational module with the application web service. If $\overline{\%\mathrm{CPU}}_N < 20\%$, the system automatically shuts down one computational module with the application web service. If $\overline{\%\mathrm{CPU}}_N$ is in the range from 20% to 50%, the system does not perform any configuration changes;
- After each change in the system's state (during scaling), a stabilization mode must be activated, during which the reactive control system does not perform any configuration changes to the application web service for a specified period of time;
- The set of Balancers is represented by a request balancing service between instances of the web application;
- When a new instance of the web application is added, it is automatically included in the load balancing. Similarly, when an instance of the web application is turned off, it is automatically excluded from the load balancing;
- The set of Clients is represented by a load testing system where you can define a workload profile for the application system based on requests and various user profiles. The workload profile can be defined using a function, pre-prepared data, or through the operation of web and CLI clients. The load testing system should display real-time statistics during testing and have the capability to save reports containing statistics on specific requests such as RPS, Response Time, errors during request execution, and the number of users;
- The load testing system operates within the computational module and can support a distributed configuration of instances running simultaneously across multiple computational modules;
- During a load test with maximum load, the CPU resource consumption by the computational module, where the load testing system operates, should not exceed the capacity of one CPU core;
- The presence of a database instance is possible but not mandatory;
- The system supports the Infrastructure as Code (IaC) model.

IV. Model System: Implementation

The algorithm used to solve the given task includes the following main stages:

- Defining the key functional blocks of the system. Preparation of a high-level schematic diagram;
- Identifying possible implementation options for each block considering the system requirements;
- Analyzing possible implementations considering the following criteria: availability of ready-made components that can be used to build functional blocks, simplicity of implementation, and implementation time;
- Choosing an implementation option for prototyping the system;
- Creating a prototype of the system;
- Qualitative assessment of the prototype's compliance with the task criteria during testing;
- If the criteria are not met, return to step 2;
- Perform the necessary number of iterations (steps 2-6) until the prototype meets the task criteria.