СОГЛАСОВАНО
Проректор по научной работе
ФГАОУ ВО "СПбПУ"
_____ В.В. Сергеев
"_____"_____ 2018

УТВЕРЖДАЮ
Первый проректор
ФГАОУ ВО "НИУ ВШЭ"
_____ Л.М. Гохберг
"_____"_____ 2018

ОПИСАНИЕ АЛГОРИТМА
ПрК ИмФ СХД

XXXX.XXXXX.XXX-ЛУ ЛИСТ УТВЕРЖДЕНИЯ

Руководитель разработки
_____ Х.Х. XXXXXXXX
"_____"_____ 2018

Ответственный исполнитель
_____ Х.Х. XXXXXXXX
"_____"_____ 2018

Москва, 2018

ФГАОУ ВО "НИУ ВШЭ"

УТВЕРЖДЕНО
ХХХХ.ХХХХХ.ХХХ-ЛУ

ОПИСАНИЕ АЛГОРИТМА
ПрК ИмФ СХД

ХХХХ.ХХХХХ.ХХХ
ЛИСТОВ YY

Содержание

1 АЛГОРИТМЫ ПрК ИмФ СХД

1.1 ОБЩИЕ СВЕДЕНИЯ

Обозначение и наименование программы

Наименование программы: «Программный комплекс имитации функционирования (ПрК ИмФ) системы хранения данных (СХД)».

Программное обеспечение, необходимое для функционирования программы

Языки программирования, на которых написана программа

Для реализации Прк ИмФ СХД используется язык программирования Go 1.10.

1.2 ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ

ПрК ИмФ СХД должен быть предназначен для имитации функционировании СХД Tatlin (показан на рис. 1) с целью проверки гипотез о состояниях СХД и переходах между ними в течение заданного интервала времени.

ПрК ИмФ СХД должен обеспечивать выполнение следующих функций:

– ввод параметров имитации;

– имитация функционирования аппаратных компонентов СХД с учетом взаимного влияния аппаратных компонентов СХД в режимах: чтения, записи и хранения данных;

– имитация функционирования СПО СХД;

– имитация функционирования СХД в целом с учетом управляющих воздействий на компоненты СХД в режимах: чтения, записи и хранения данных;

– имитация нагрузки на СХД.

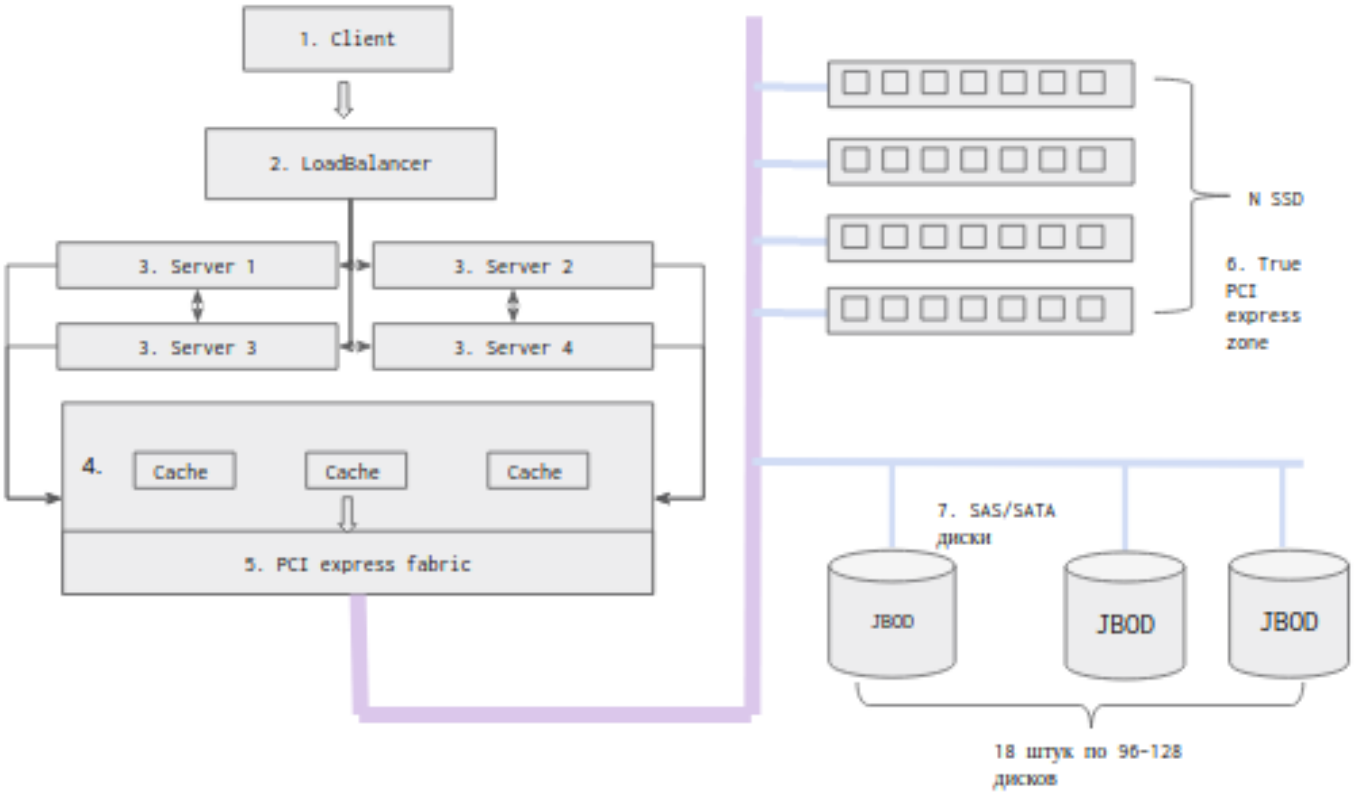

Рисунок 1 — СХД Татлин.

1.3 ОПИСАНИЕ ЛОГИЧЕСКОЙ СТРУКТУРЫ

Алгоритм программы

Используемые методы

1.4 Описание окружения имитации

Имитация системы происходит при помощи объекта типа Environment, который полностью отражает текущее положение системы. Данный объект обладает следующими полями:

currentTime Текущее время системы типа $float64$. Изменяется дискретными шагами.

workers Словарь типа $map[uint64] * Process$, где в качестве ключа выступает идентификатор PID объекта, а в качестве значение указатель $Worker$.

routesMap Словарь типа $map[Route] * Link$. Содержит значения обо всех путях, возможной в данной конфигурации компьютерной сети. $Route$ – структура, имеющая в качестве своих полей $start$ и $finish$ – указатели на хост, которые

являются началом и концом пути, соответственно. Значение $*Link$ – является указателем на сеть, по которой будет проходить передача пакетов в обе стороны.

queue Поле типа *eventQueue*. Является глобавльным хранилищем всех событий во время имитации сложных сложных систем.

mutex Поле типа $sync.Mutex$. Примитив синхронизации нужный для того, чтобы обеспечивать консистентность доступа к данным, таким как очередь событий *queue*.

shouldStop Поле типа *bool*. Во время прогона симуляции является равным 1. После того, как все события в имитации заканчиваются, либо при наличии специального события, выставляется в отрицательное значение и прогон прекращается.

hostsMap Словарь типа $map[string]HostInterface$. Содержит информацию обо всех хостах, которые имеются в симуляции. В качестве ключа словаря – имя хоста, в качестве значения интерфейс типа HostInterface, который обобщает такие типы как *host*, $NetworkSwitch$ и $IOBalancer$.

vesninServers Поле типа $[] * Host$. Является списком, который содержит информацию о рабочих дисковых контроллерах, поддреживающих отношения с клиентом, представленных в виде указателя на $*Host$.

allVesninServers Поле типа $[] * Host$. Поле типа $[] * Host$. Является списком, который содержит информацию о обо всех (рабочих и нерабочих) дисковых контроллерах, поддреживающих отношения с клиентом, представленных в виде указателя на $*Host$.

storagesMap Словарь типа $map[string] * Storage$. В данном контейнере хранится информация обо всех дисках, примонтированных к системе хранения данных. В качестве ключа словаря используется идентификатор конечного хранилища данных, а качестве значения – указатель на дисковое хранилище.

linksMap Словарь типа $map[string] * Link$. Контейнер, содержащий информацию обо всех сетях, представленных в данной симуляции. В качестве ключа словаря используется идентификатор сети, а качестве значения – сеть, по которой будет идти передача данных.

FunctionsMap Словарь типа $map[string]func(*Process, []string)$. Данный контейнер хранит информацию обо всех фукнциях, которые будует запущены в качестве горутин в начальный момент времени (время запуска симуляции). Функ-

ции должны быть объявлены в файле deployment.xml. В качестве ключа словаря используется идентификатор функции, представленный в строковом виде, а качестве значения – указатель на функцию.

daemonList Поле типа $[]*Process$. Является списком, который содержит информацию о горутинах, которые во время симуляции СХД, являются представлениями Unix-демонов и самостоятельно должны завершить своё исполнение.

pid Поле типа $ProcessID$. Указатель на функцию, которая исполняется в текущий момент времени.

waitWorkerAmount Поле типа $uint64$. Количество горутин, запущенных в текущий момент времени, завершения которых нужно ожидать для того, чтобы наполнить очередь актуальными текущими событиями.

stepEnd Поле типа $chaninterface$. Является средством коммукации горутин с главной ($master$) горутиной. В данный канал связи горутины, которые исполняются в текущий момент времени, сигнализируют о своём завершении.

nextWorkers Поле типа $[]*Process$. Является списком, который содержит информацию о горутинах, которые должны буть запущены на следующем шаге работы имитации системы со статусом OK.

timeOutWorkers Поле типа $[]*Process$. Является списком, который содержит информацию о горутинах, которые должны буть запущены на следующем шаге работы имитации системы со статусом $TIMEOUT$.

anomalyWorkers Поле типа $[]*Process$. Является списком, который содержит информацию о горутинах, которые должны буть запущены на следующем шаге работы имитации системы со статусом $FAIL$.

logsMap Словарь типа $map[string]float64$. Данный контейнер хранит информацию, которая впоследствии будет выведена в виде логов системы. В качестве ключа словаря используется идентификатор наблюдаемого значения, а качестве значения – числовая характеристика данной величины.

unitsMap Словарь типа $map[string]float64$. Данный контейнер хранит информацию о единицах системы измерений, принятых в данной симуляции. В качестве ключа словаря используется идентификатор единицы измерения, а качестве значения – численная характерстика относительно эталона.

backupRoutesMap Словарь типа $map[Route]*Link$. Содержит значения обо всех запасных (backup) путях, возможной в данной конфигурации компью-

терной сети. $Route$ – структура, имеющая в качестве своих полей $start$ и $finish$ – указатели на хост, которые являются началом и концом пути, соответственно. Значение $*Link$ – является указателем на сеть, по которой будет проходить передача пакетов в обе стороны.

HostLinksMap Словарь типа $map[HostInterface][] * Link$. Данный контейнер хранит информацию о сетях, к которым имеет доступ каждый хост. В качестве ключа словаря используется идентификатор хоста, а качестве значения – список, состоящий из указателей на сеть, принадлежащих даннному хосту.

LinkBackupsMap Словарь типа $map[*Link] * Link$. В качестве ключа словаря используется идентификатор конечного хранилища данных, а качестве значения – указатель на дисковое хранилище.

Функции Environment

Дискретно-событийная симуляция моделирует работу системы как дискретную последовательность событий во времени. Каждое событие происходит в определенный момент времени и отмечает изменение состояния в системе. Между последовательными событиями никаких изменений в системе не предполагается; таким образом, симуляция может непосредственно переходить во времени от одного события к другому. Схема данного действия изображена на рис. 2.

func NewEnvironment() *Environment Входные аргументы: отсутствуют.

–
–
–

Выходное значение: переменная типа *Environment. Описание функции:Создаёт и инициализует необходимые поля для функцирования симуляции, такие как:

- queue
- workers
- SendEventsNameMap
- ReceiveEventsNameMap
- ReceiverSendersMap
- stepEnd

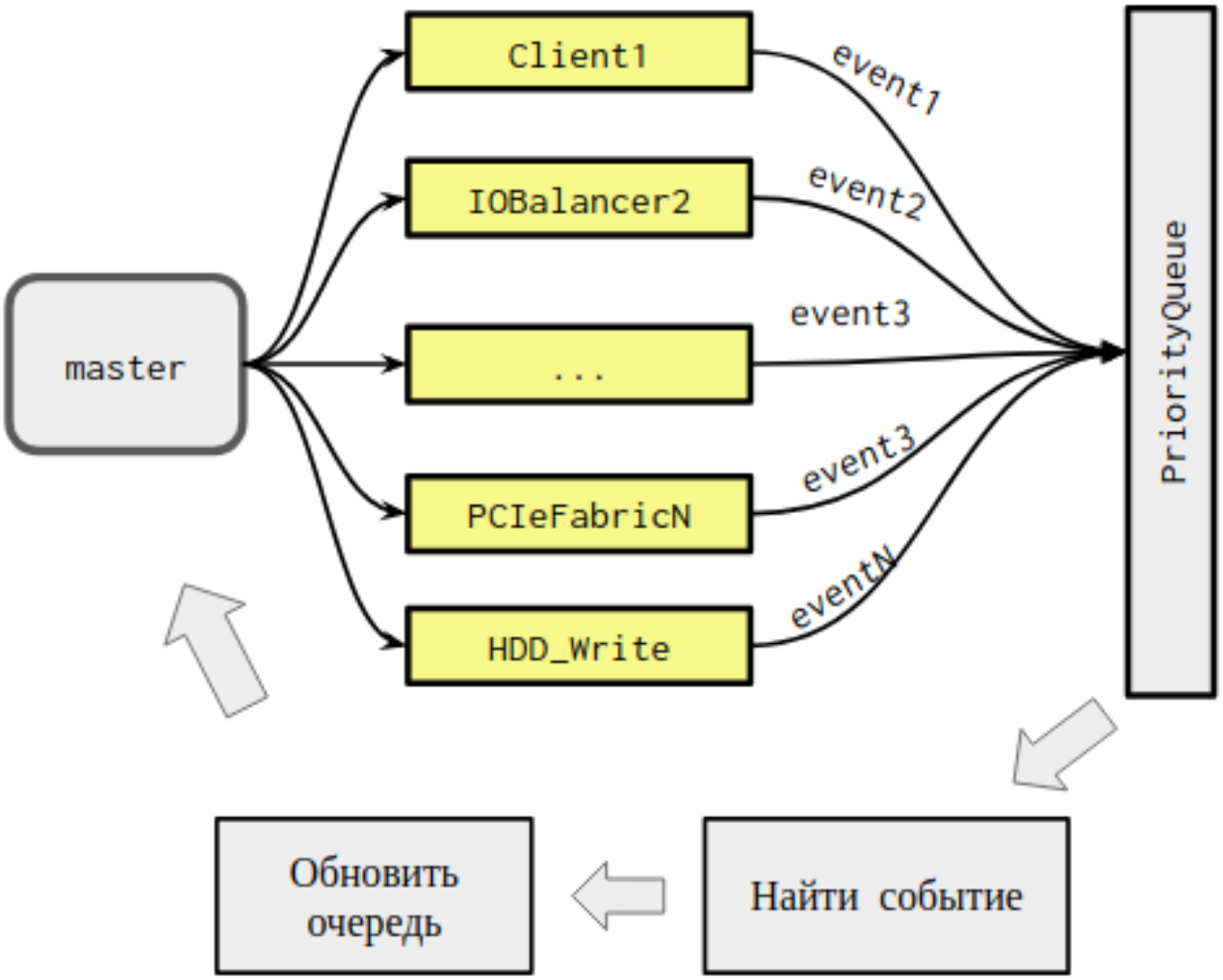

Рисунок 2 — Один цикл симуляции

– logsMap
– HostLinksMap
– LinkBackupsMap

func createUnits()

Входные аргументы: отсутствуют.

Выходное значение: отсутствует.

Описание функции: Инциализирует единицы измерения необходимые при симуляции системы (показано в таблице 11tab:SI).

Таблица 1 — Единицы системы измерений при моделировании

TB	1000^4 byte
GB	1000^3
MB	1000^2
KB	1000

Выходное значение: отсутствует.

Описание: Данная функция обновляет очередь событий за время $deltaTime$.

func (env *Environment) CreateTransferEvents()

Входные аргументы: указатель на объект типа *Environment.

–

Выходное значение: отсутствует.

Описание: Данная функция создаёт события, которые имитируют передачу данных от одного хоста к другому.

func (env *Environment) Step() EventInterface

Входные аргументы: указатель на объект типа *Environment.

–

Выходное значение: текущее событие симуляции.

Описание: Данная функция осуществляет шаг симуляции, который состоит из следующих шагов.

а) Создать события, которые имитируют передачу данных от одного хоста к другому.

б) Проверить является ли этот шаг симуляции последним.

в) Проверить симуляцию на возникновение дедлоков.

г) Получить событие из очереди с минимальным значением времени.

д) Обновить текущее время.

е) Обновить очередь событий за время, прошедшее с времени прошлого события.

ж) Обработать коллбэки (callbaks) текущего события.

з) Проверить является ли этот шаг симуляции последним.

func (env *Environment) FindNextWorkers(event EventInterface)

Входные аргументы: указатель на объект типа *Environment, текущее событие EventInterface.

–

Выходное значение: отсутствует.

Описание: Данннная функция занимается поиском горутин, которые должны начать исполнение после выполнения текущего шага. Данный список включает в себе также горутины, которые начнут исполнение со статусами OK, $FAIL$, $TIMEOUT$.

func (env *Environment) SendStartToSignalWorkers()

Входные аргументы: указатель на объект типа *Environment.

–

Выходное значение: отсутствует.

Описание: Данннная функция рассылает сигналы через каналы коммуикации горутинам, которые должны начать исполнение после выполнения текущего шага. Данный список включает в себе также горутины, которые начнут исполнение со статусами OK, $FAIL$, $TIMEOUT$.

func (env *Environment) WaitWorkers()

–

Входные аргументы: указатель на объект типа *Environment.

Выходное значение: отсутствует.

Описание: Данннная функция дожидается выполнения задач текущими горутинами, которым были посланы сигналы на предыдущем этапе.

1.5 Описание примитивов, использованных при имитации системы

Имитация сети

Компьютерная сеть или сеть передачи данных - это цифровая телекоммуникационная сеть, которая позволяет узлам совместно использовать ресурсы. В компьютерных сетях вычислительные устройства обмениваются данными друг с другом с использованием соединений между узлами (линией передачи данных). Эти линии передачи данных устанавливаются на кабельных носителях, таких как провода или оптические кабели. Сетевые компьютерные устройства, которые создают, маршрутизируют и завершают данные, называются сетевыми узлами. Узлы могут включать хосты, PCIe-фабрику, а также сетевое оборудование. Можно сказать, что два таких устройства объединены в сеть, когда одно устройство мо-

жет обмениваться информацией с другим устройством, независимо от того, имеет ли оно прямое соединение друг с другом.

Сеть имитируется при помощи структуры *Link*. Она обладает следующими полями (характеристиками).

name Идентификатор сети в текстовом представлении типа string.

state float64 Степень соответсвия изначальному ресурсу, либо 1 минус деградация данной сети. Значение типа float64, может принимать значения от 0 до 1, где 0 соответсвует полной деградации сети, а 1 – "фабричному"состоянию.

route *Route Указатель на структуру данных Route, которая содержит информацию о хостах, которые соединяет данная сеть.

minEvent Указатель на минимальное событие-пакет *TransferEvent, которое передаётся в текущий момент по сети.

bandwidth Переменная типа float64. Пропускная способность сети, которая изменяется в байтах в секунду.

lastTimeRequest Переменная типа float64. Время последнего обращения к данной сети.

mutex Примитив синхронизации типа sync.Mutex необходимой для корректности парального доступа к полям структуры данной сети.

counter Переменная типа int64. Количество пакетов, которые передаются в текущий момент времени по сети.

func NewLink(bandwidth float64, name string) *Link

Входные аргументы: bandwidth – переменная типа float64. Содержит информацию о пропускной способности сети. name – переменная типа string, имя сети.

Выходное значение: Указатель созданную структуру, которая инкапсулирует сеть.

Описание функции: Создаёт указатель созданную структуру, которая инкапсулирует сеть с именем name и пропускной способностью bandwidth и инициализирует необходимые поля сети, такие как:

- bandwidth
- mutex
- name
- state

func (link *Link) Put(e *TransferEvent)

Входные аргументы: Указатель на структуру *Link, указатель на событие, которое должно передаваться по сети.

–

Выходное значение: отсутствует.

Описание функции: Данная функция добавляет событие в очередь событий, относящейся к сети link.

func (link *Link) EstimateTimeEnd(e *SendEvent)

Входные аргументы: Указатель на структуру Link; указатель на событие SendEvent, которое должно передаваться по сети.

–

Выходное значение: отсутствует

Описание функции: Оценить время окончания t_{end} передачи события-пакета по данной сети по следующей формуле:

$$t_{end} = t_0 + \frac{S}{\frac{B}{n} \cdot q}$$

,

где t_0 – это текущее время, S – размер передаваемого пакета, B – пропускная способность сети, n – количество пакетов, которые передаются в текущий момент времени, q – степень деградации сети.

func (env *Environment) FindNextTransferEvent()

–

Входные аргументы: Указатель на структуру *Environment.

Выходное значение: Отсутствует.

Описание функции: Данная функция "составляет"события, которые будут передаваться в текущий момент времени.

func GetRoute(route Route) *Link

Входные аргументы: route переменна типа Route, содержащая информацию об начальном и конечном хостах.

–

Выходное значение: Указатель на структуру Link.

Описание функции: Данная функция по имени route возвращает указатель на структуру Link.

Route обладает следующими полями.

Указатель на начальный start. Тип HostInterface

Указатель на конечный finish. Тип HostInterface

Имитация хоста

Сетевой хост - это компьютер или другое устройство, подключенное к компьютерной сети. Сетевой хост может предоставлять информационные ресурсы, службы и приложения пользователям или другим узлам в сети. Сетевой узел - это сетевой узел, которому назначен сетевой адрес. Компьютеры, участвующие в сетях, которые используют пакет интернет-протокола, также могут называться IP-узлами. В частности, компьютеры, участвующие в Интернете, называются интернет-хостами, иногда интернет-узлами. Интернет-хосты и другие IP-хосты имеют один или несколько IP-адресов, назначенных их сетевым интерфейсам. Адреса настраиваются либо вручную администратором, либо автоматически. В общем случае, все серверы - это хосты, но не все хосты - это серверы. Любое устройство, установившее соединение с сетью, квалифицируется как хост, тогда как только хосты, которые принимают подключения от других устройств (клиентов), квалифицируются как серверы.

–

name Поле типа string. Является идентификатором объекта.

typeId Поле типа string. Содержит информацию о классе устройств, которым принадлежит данный хост.

processes Поле типа []*Process. Содержит информацию в виде списка указателей на Process обо всех текущих процессах, запущенных на данном хосте.

speed Поле типа float64. Скорость работу данного хоста, измеряемая в flops.

storage Поле типа *Storage. Содержит указатель на диск, который примонтирован к данному хосту.

traffic Поле типа float64. Траффик в байт/с, который проходит через данный хост.

logs Поле типа interface{}. Текстовове представление логов данного хоста.

Функции необходимые для имитации хоста

func (env *Environment) getHostByName(name string) HostInterface

Входные аргументы: Аргумент name типа string.

–

Выходное значение: Объект типа HostInterface.

Описание функции: Данная функция по данному имени name возвращает объект типа HostInterface.

func (process *Process) GetHost() HostInterface

Входные аргументы: Указатель на процесс, владеющий в данное время исполнением.

–

Выходное значение: Объект типа HostInterface

Описание функции: Данная функция возвращает хост HostInterface, на котором в данное время исполняется текуща горутина.

func (host *Host) GetName() string

Входные аргументы: Указатель на объект Host.

Выходное значение: Идентификатор хоста тип string.

Описание функции: Данная функция возвращает имя текущего хоста.

func (host *Host) GetType() string

Входные аргументы: Указатель на объект Host.

–

Выходное значение: Тип класса устройств к которым относится данный хост. Текстовое представление.

Описание функции: Данная функция возвращает тип текущего хоста.

func (host *Host) GetDevTemp() float64

Входные аргументы: Указатель на объект Host.

–

Выходное значение: Температура данного хоста. Тип float64.

Описание функции: Данная функция возвращает температуру данного хоста в текущий момент времени.

func (host *Host) GetTraffic() float64

Входные аргументы: Указатель на объект Host.

–

Выходное значение: Суммарный (входной и выходной) траффик, проходящий, через данный хост.

Описание функции: Данная функция возвращает значение суммарного (входного и выходного) траффика, проходящего, через данный хост.

func (host *Host) AddTraffic(traffic float64)

Входные аргументы: Указатель на объект Host.

–

Выходное значение: Отсутствует.

Описание функции: Данная функция кумулятивно увеличивает суммарное значение выходного траффика на значение traffic.

func (host *Host) GetLoad() int

Входные аргументы: Указатель на объект Host.

–

Выходное значение: Загрузка процессора в текущий момент времени.

Описание функции: Данная функция возвращает загрузку процессора в текущий момент времени.

func (host *Host) GetLogs() interface

Входные аргументы: Указатель на объект Host.

–

Выходное значение: Логи компоненты системы.

Описание функции: Данная функция возвращает логи компоненты системы.

func (host *Host) SetLogs(logs interface)

Входные аргументы: Указатель на объект Host.

–

Выходное значение: Отсутствует.

Описание функции: Обновляет логи текущей компоненты системы.

func (host *Host) GetStorage() *Storage

Входные аргументы: Указатель на объект Host.

–

Выходное значение: Указатель на объект Storage, объект симулирующий конечный дисковый носитель.

Описание функции: Данная функция возвращает указатель на объект Storage, объект симулирующий конечный дисковый носитель.

func GetHostByName(hostName string) HostInterface

Входные аргументы: Строка – имя запрашиваемого хоста.

–

Выходное значение: Указатель на объект Host.

Описание функции: Данная функция возвращает указатель на объект Host по его строковому указателю.

Имитация конечного дискового хранилища

–

StorageType Структура данных StorageType необходима при модировании класса конечных дисковых носителей. Данная структура обладает следующими полями:

typeId. Тип string. Идентификатор класса, к которому принадлежит данный тип конечных дисковых носителей. **writeRate**. Тип float64. Скорость данных на запись конечного дискового носителя. **readRate** . Тип float64. Скорость данных на чтение конечного дискового носителя. **size** . Тип float64. Размер конечного дискового носителя.

Конкретная реализация конечного дискового носителя осуществляеся при помощи примитива Storage. Он обладает следующими полями.

***StorageType**. Указатель на класс устройств конечного дискового носителя. **name** . Тип string. Идентификатор конечного дискового носителя. **readLink** . Тип *Link. Указатель структуру, по которой происходит запись на конечный дисковый носитель. **writeLink** . Тип *Link. Указатель структуру, по которой происходит чтение на конечный дисковый носитель.

usedSize . Тип int64. Занятое место на конечном дисковом носителе. **logs** . Тип interface. Логи, принадлежащие конечному дисковому носителю.

func NewStorage(storageType *StorageType, name string) *Storage

Входные аргументы: Тип конечного носителя storageType, имя, создаваемого объекта, name.

–

Выходное значение: Указатель на объект Storage.

Описание функции: Данная функция создаёт объект, имитирующий поведение конечного диского носителя.

func GetDiskDrives() map[string]*Storage

Входные аргументы: отсутствуют.

Выходное значение: Словарь, где в качестве ключа используется строка, а качестве значения конечный дисковый носитель, имеющий такое же имя.

Описание функции: Данная функция возвращет словарь, содержащий сведения обо всех дисковых носителях, имеющихся в конкретной реализации.

func (storage *Storage) GetLogs() interface

Входные аргументы: Указатель на структуру, имитирующую конечный дисковый накопитель.

–

Выходное значение: Логи дискового компонента.

Описание функции: Данная функция возвращает логи дисковой компоненты.

func (storage *Storage) SetLogs(logs interface)

Входные аргументы: Указатель на структуру, имитирующую конечный дисковый накопитель.

–

Выходное значение: Отсутствует.

Описание функции: Данная функция обновляет логи дисковой компоненты.

func (storage *Storage) GetName() string

Входные аргументы: Указатель на структуру, имитирующую конечный дисковый накопитель.

–

Выходное значение: Имя конечного дискового накопителя.

Описание функции: Данная функция имя конечного дискового накопителя.

func (storage *Storage) GetDevTemp() float64

Входные аргументы: Указатель на структуру, имитирующую конечный дисковый накопитель.

–

Выходное значение: Температура конечного дискового накопителя.

Описание функции: Данная функция возвращает температуру конечного дискового накопителя.

func (storage *Storage) GetRawCapacity() float64

Входные аргументы: Указатель на структуру, имитирующую конечный дисковый накопитель.

Выходное значение: Заявленная ёмкость конечного дискового накопителя.

Описание функции: Данная функция возвращает ёмкость конечного дискового накопителя.

func (storage *Storage) GetAvgReadSpeed() float64

Входные аргументы: Указатель на структуру, имитирующую конечный дисковый накопитель.

–

Выходное значение: Средняя скорость на чтение конечного дискового накопителя.

Описание функции: Данная функция возвращает среднюю скорость на чтение конечного дискового накопителя.

func (storage *Storage) GetAvgWriteSpeed() float64

Входные аргументы: Указатель на структуру, имитирующую конечный дисковый накопитель.

–

Выходное значение: Средняя скорость на запись конечного дискового накопителя.

Описание функции: Данная функция возвращает среднюю скорость на запись конечного дискового накопителя.

func (storage *Storage) GetDataInterfaceCNT() uint16

Входные аргументы: Указатель на структуру, имитирующую конечный дисковый накопитель.

–

Выходное значение: Интерфейс передачи данных.

Описание функции: Данная функция возвращает интерфейс передачи данных.

func (storage *Storage) GetUsedSpace() float64

Входные аргументы: Указатель на структуру, имитирующую конечный дисковый накопитель.

–

Выходное значение: Размер занятого места на конечном дисковом накопителе.

Описание функции: Данная функция возвращает размер занятого места на конечном дисковом накопителе.

func (storage *Storage) GetFreeSpace() float64

Входные аргументы: Указатель на структуру, имитирующую конечный дисковый накопитель.

–

Выходное значение: Размер свободного места на конечном дисковом накопителе.

Описание функции: Данная функция возвращает размер свободного места на конечном дисковом накопителе.

func (storage *Storage) WritePacketSize()

Входные аргументы: Указатель на структуру, имитирующую конечный дисковый накопитель.

–

Выходное значение: Отсутствует.

Описание функции: Данная функция имитирует запись на конечный дисковый накопитель одного пакета данных.

func (storage *Storage) DeletePacketSize()

Входные аргументы: Указатель на структуру, имитирующую конечный дисковый накопитель.

–

Выходное значение: Отсутствует.

Описание функции: Данная функция имитирует удаление с конечного дискового накопителя одного пакета данных.

Имитация коммутатора внутренней управляющей сети СХД

NetworkSwitch Структура данных NetworkSwitch необходима при моделировании коммутатора внутренней управляющей сети СХД. Данная структура обладает следующими функциями:

func(ns *NetworkSwitch) GetConnectedDevCnt() uint8

Входные аргументы: Указатель на структуру, имитирующую коммутатор внутренней управляющей сети СХД.

–

Выходное значение: Количество подключенных устройств.

Описание функции: Данная функция возвращает количество подключенных устройств.

func(ns *NetworkSwitch) GetOnlineSwitchesCnt() uint8

Входные аргументы: Указатель на структуру, имитирующую коммутатор внутренней управляющей сети СХД.

Выходное значение: Количество одновременно включенных коммутаторов.

Описание функции: Данная функция возвращает количество одновременно включенных коммутаторов.

Имитация фабрики PCI Express

PCIeFabric Структура данных PCIeFabric необходима при моделировании фабрики PCI Express. Данная структура обладает следующими функциями:

func(pc *PCIeFabric) GetPCIeDevicesCnt() uint16

Входные аргументы: Указатель на структуру, имитирующую фабрику PCI Express.

Выходное значение: Количество подключенных устройств.

Описание функции: Данная функция возвращает количество подключенных устройств.

func(pc *PCIeFabric) GetPCIeMaxBandw() float64

Входные аргументы: Указатель на структуру, имитирующую фабрику PCI Express.

Выходное значение: Максимальная пропускная способность.

Описание функции: Данная функция возвращает значение максимальной пропускной способности.

Имитация балансировщика нагрузки

IOBalancer Структура данных IOBalancer необходима при моделировании балансировщика нагрузки. Данная структура обладает следующими полями:

***Host**. Наследование от базового типа Host, т.е помимо нижеперечисленных полей, тип IOBalancer обладает также и полями типа Host. **readResponseTime**. Тип float64. Время отклика на запросы чтения. **writeResponseTime**. Тип float64. Время отклика на запросы записи. **writeRequests**. Тип float64. Количество запросов на запись в единицу времени. **readRequests**. Тип float64. Количество запросов на чтение в единицу времени. **lastTime**. Тип float64. Время последнего запроса.

func(iob *IOBalancer) GetReadRequestsRate() float64

Входные аргументы: Указатель на структуру, имитирующую балансировщик нагрузки.

–

Выходное значение: Количество обработанных запросов на чтение в секунду.

Описание функции: Данная функция возвращает количество обработанных запросов на чтение в секунду.

func(iob *IOBalancer) GetWriteRequestsRate() float64

Входные аргументы: Указатель на структуру, имитирующую балансировщик нагрузки.

–

Выходное значение: Количество обработанных запросов на запись в секунду.

Описание функции: Данная функция возвращает количество обработанных запросов на запись в секунду.

func(iob *IOBalancer) GetReadResponseDelay() float64

Входные аргументы: Указатель на структуру, имитирующую балансировщик нагрузки.

–

Выходное значение: Время отклика при запросе на чтение.

Описание функции: Данная функция возвращает значение времени отклика при запросе на чтение.

func(iob *IOBalancer) GetWriteResponseDelay() float64

Входные аргументы: Указатель на структуру, имитирующую балансировщик нагрузки.

–

Выходное значение: Время отклика при запросе на запись.

Описание функции: Данная функция возвращает значение временпи отклика при запросе на запись.

func(iob *IOBalancer) GetReadDataVolume() float64

Входные аргументы: Указатель на структуру, имитирующую балансировщик нагрузки.

–

Выходное значение: Объем передаваемых данных в режиме на чтение.

Описание функции: Данная функция возвращает объем передаваемых данных в режиме на чтение.

func(iob *IOBalancer) GetWriteDataVolume() float64

Входные аргументы: Указатель на структуру, имитирующую балансировщик нагрузки.

–

Выходное значение: Объем передаваемых данных в режиме на запись.

Описание функции: Данная функция возвращает объем передаваемых данных в режиме на запись.

func(iob *IOBalancer) GetReadRequestProcessTime() float64

Входные аргументы: Указатель на структуру, имитирующую балансировщик нагрузки.

–

Выходное значение: Среднее время обработки запросов на чтение.

Описание функции: Данная функция возвращает среднее значение времени обработки запросов на чтение.

func(iob *IOBalancer) GetWriteRequestProcessTime() float64

Входные аргументы: Указатель на структуру, имитирующую балансировщик нагрузки.

–

Выходное значение: Среднее время обработки запросов на запись.

Описание функции: Данная функция возвращает возвращает среднее значение времени обработки запросов на запись.

func(iob *IOBalancer) GetIoProcessingMethod() uint8

Входные аргументы: Указатель на структуру, имитирующую балансировщик нагрузки.

–

Выходное значение: Способ обработки операций ввода-вывода (синхронный/асинхронный).

Описание функции: Данная функция возвращает способ обработки операций ввода-вывода (синхронный/асинхронный).

func(iob *IOBalancer) GetReadCancelRate() float64

Входные аргументы: Указатель на структуру, имитирующую балансировщик нагрузки.

–

Выходное значение: Количество аннулированных запросов на чтение в единицу времени.

Описание функции: Данная функция возвращает количество аннулированных запросов на чтение в единицу времени.

func(iob *IOBalancer) GetWriteCancelRate() float64

Входные аргументы: Указатель на структуру, имитирующую балансировщик нагрузки.

–

Выходное значение: Количество аннулированных запросов на запись в единицу времени.

Описание функции: Данная функция возвращает количество аннулированных запросов на запись в единицу времени.

func(iob *IOBalancer) GetBlockSize() uint16

Входные аргументы: Указатель на структуру, имитирующую балансировщик нагрузки.

–

Выходное значение: Размер блока данных при чтении/записи.

Описание функции: Данная функция возвращает размер блока данных при чтении/записи.

func(iob *IOBalancer) GetIOProcessCnt() uint8

Входные аргументы: Указатель на структуру, имитирующую балансировщик нагрузки.

–

Выходное значение: Количество процессов, генерирующих запросы на чтение/запись.

Описание функции: Данная функция возвращает количество процессов, генерирующих запросы на чтение/запись.

func(iob *IOBalancer) GetReadQueueLength() uint8

Входные аргументы: Указатель на структуру, имитирующую балансировщик нагрузки.

Выходное значение: Длина очереди запросов на чтение для асинхронных операций ввода-вывода.

Описание функции: Данная функция возвращает длину очереди запросов на чтение для асинхронных операций ввода-вывода.

func(iob *IOBalancer) GetWriteQueueLength() uint8

Входные аргументы: Указатель на структуру, имитирующую балансировщик нагрузки.

–

Выходное значение: Длина очереди запросов на запись для асинхронных операций ввода-вывода.

Описание функции: Данная функция возвращает длину очереди запросов на запись для асинхронных операций ввода-вывода.

func(iob *IOBalancer) SetReadResponseDelay(rt float64

Входные аргументы: Указатель на структуру, имитирующую балансировщик нагрузки; время отклика запросов на чтение rt.

Выходное значение: Отсутствует

Описание функции: Данная функция устанавливает время отклика при запросах на чтение.

func(iob *IOBalancer) SetWriteResponseDelay(rt float64

–

Входные аргументы: Указатель на структуру, имитирующую балансировщик нагрузки; время отклика запросов на запись rt.

Выходное значение: Отсутствует.

Описание функции: Данная функция устанавливает время отклика при запросах на чтение.

Имитация задач в симуляторе

Task Структура данных Task необходима при моделировании задач в симуляторе. Данная структура обладает следующими полями, характеризующими её:

name. Тип string. Идентификатор задачи.

size. Тип float64. Размер задачи в байтах. Необходим в случае передачи данных по сети или сохранении задачи на диск.

flops. Тип float64. Вычислительная сложность задачи во флопсах. Необходима для оценки времени, которое затратит хост при обработке данной задачи.

data. Тип interface. Ссылка на дополнительную структуру данных, которыми может обладать данная задача.

Соответственно, данный тип данных обладает следующим набором функций:

func NewTask(name string, flops float64, size float64, data interface) *Task

Входные аргументы: имя задачи, вычислительный размер задачи, размер задачи, ссылка на дополнительную структуру данных.

–

Выходное значение: Объект типа Task.

Описание функции: Данная функция создаёт задачу с параметрами, указанными во входных аргументах.

func (task *Task) GetName() string

Входные аргументы: Отсутствуют.

Выходное значение: Идентификатор задачи.

Описание функции: Данная функция возвращает идентификатор задачи.

func (task *Task) GetSize() float64

Входные аргументы: Отсутствуют.

Выходное значение: Размер задачи.

Описание функции: Данная функция возвращает размер задачи.

func (task *Task) GetFlops() float64

Входные аргументы: Отсутствуют.

Выходное значение: Вычислительный размер задачи.

Описание функции: Данная функция возвращает вычислительный размер задачи.

func (task *Task) GetData() interface

Входные аргументы: Отсутствуют.

Выходное значение: Ссылка на дополнительную структуру данных, которыми может обладать данная задача.

Описание функции: Данная функция возвращает ссылка на дополнительную структуру данных, которыми может обладать данная задача.

func (process *Process) Execute(task *Task)

Входные аргументы: Задача, которую необходимо обработать.

–

Выходное значение: Статус выполения задачи.

Описание функции: Данная функция имитирует поведение СХД при исполнении задачи.

Структура Process обладает следующими методами.

func (pid *ProcessID) Next() uint64

Входные аргументы: Указатель на структуру, которая симулирует процесс.

–

Выходное значение: Идентификатор объекта.

Описание функции: Данная функция генерирует новый уникальный идентификатор для процесса.

func ProcWrapper(processStrategy func(*Process, []string), w *Process, args []string)

Входные аргументы: Указатель на структуру, которая симулирует процесс, указатель на функцию, которая будет симулировать стратегию, которая реализовавается на реальном хосте, набор аргументов, которым обладает хост.

–

Выходное значение: Процесс.

Описание функции: Данная функция по указанным входным параметрам создаёт и запускает процесс.

func (process *Process) Daemonize()

Входные аргументы: Указатель на структуру, которая симулирует процесс.

–

Выходное значение: Отсутствует.

Описание функции: Данная функция превращает текущий процесс в процесс-демон.

func (p *Process) GetData() interface

Входные аргументы: Указатель на структуру, которая симулирует процесс.

–

Выходное значение: Указатель на структуру данных о дополнительных параметрах, которыми обладает хост.

Описание функции: Данная функция возвращает указатель на струку данных о дополнительных параметрах, которыми обладает хост.

func (p *Process) GetName() string

Входные аргументы: Указатель на структуру, которая симулирует процесс.

Выходное значение: Имя процесса.

Описание функции: Данная функция имя процесса.

func (p *Process) GetEnv() *Environment

Входные аргументы: Указатель на структуру, которая симулирует процесс.

Выходное значение: Объект, указывающий на окружение имитации.

Описание функции: Данная функция возвращает объект, указывающий на окружение имитации.

Имитация процессов

Процесс представляет собой экземпляр выполнения какой-либо логики последовательности действий в симуляции. Он содержит программный код и его текущую деятельность.

–

Process Структура данных Process необходима при моделировании процессов в симуляторе. Данная структура обладает следующими полями, характеризующими её:

pid. Тип uint64. Идетификатор процесса.

env. Тип *Environment. Ссылка на Environment, в котором осуществляется симуляция.

resumeChan. Тип chan STATUS. Канал по которому осуществляется взаимодействие главной горутины с данной горутиной.

host. Тип HostInterface. Хост на котором существует данный процесс.

name. Тип string. Имя данного процесса.

noMoreEvents. Тип bool. Булева переменная, которая выставляется в положительное значение, при завершение работы процесса.

data. Тип interface. Ссылка на структуру, в которой может содержаться дополнительная информация о данном процессе.

Done. Тип chan struct. Канал по которому данный процесс сообщает мастерской горутине о своём завершении.

1.6 Устройство очереди

Наиболее важным элементом при реализации моделирования сложных систем является поддержание консистентности очереди событий. В текущей версии библиотеки она реализована при помощи встроенного в язык программирования интерфейса "container/heap". Данный интерфейс представляет структуру данных под названием дерево, которое обладает свойством, что каждая его узел является минимальным значением в его поддереве. Эта структура данных была выбрана для моделирования, т.к является наиболее распространенной при реализации очереди событий с приоритетом, которым в случае моделирования событийных имитаций является время окончания события.

Для имплементации данного интерфейса были реализованы следующие функции, которые показаны в таблице 3.

Таблица 3 — Функции необходимые для реализации на очереди

Функция	Входные аргументы	Выходные аргументы	Описание функции
func (eq eventQueue) Len() int	-	Длина очереди, тип int	Данная функция возвращает значение длины очереди.
func (eq eventQueue) Less(i, j int) bool	Индексы элементов очереди. Тип int	Тип bool	Данная функция задаёт правило сравнения и сравнивает элемент очереди с индексом i с элементом с индексом j. В случае, если первый элемент больше, то возвращается логическое да, в противном случает – логическое нет.
func (eq eventQueue) Swap(i, j int)	Индексы элементов очереди. Тип int	-	Данная функция меняет местами элемент очереди с индексом i с элементов очереди с идексом j.
func (eq *eventQueue) Push(e interface{})	Значение, которое необходимо добавить в очередь. Тип interface{}	-	Данная функция добавляет новый элемент e в очередь событий.
func (eq *eventQueue) Pop() interface{}	-	Минимальнй элемент в очереди. Тип interface{}	Данная функция извлекает минимальный эле-

Структура алгоритма с описанием функций составных частей

Преобразование входных данных

Моделирование сбоев

Преобразование выходных данных

Связь программы с другими программами

ПрК ИмФ в ходе прогона имитации СХД каждый дискретный шаг времени, который может настраиваться, посылает сообщения в виде JSON-объектов подсистеме стохастической имитационной дискретно-событийная модели функционирования аппаратных компонентов системы хранения данных (ПИДС ФАК СХД). Данный сервис по поступающим сообщениям, в которых содержится информация о состоянии имитации СХД в текущий момент времени, принимает решение о настройке внутренних параметров симулятора, таких как:

- Сети
  - Эффективная пропускная способность
  - Задержка сети
- Процессоры
  - Вычислительный ресурс
- Диски
  - Скорость на чтение
  - Скорость на запись

Кроме тюнинга параметров симулятора, ПИДС ФАК СХД также генерирует события, которые отвечают аномалиям у процессоров и сетей. Данные события отправляются в виде обратных сообщений ПрК ИмФ. Результаты работы симулятора сохраняются в подсистеме хранения данных (ПХД). Алгоритм предсказания сбоев (АПС СХД), взаимодействия с ПХД, получает данные симулятора для проведения корреляционного анализа, анализа временных рядов и построения искусственных нейронных сетей. Схема на рисунке 3 показывает взаимодействие вышеперечисленных компонент.

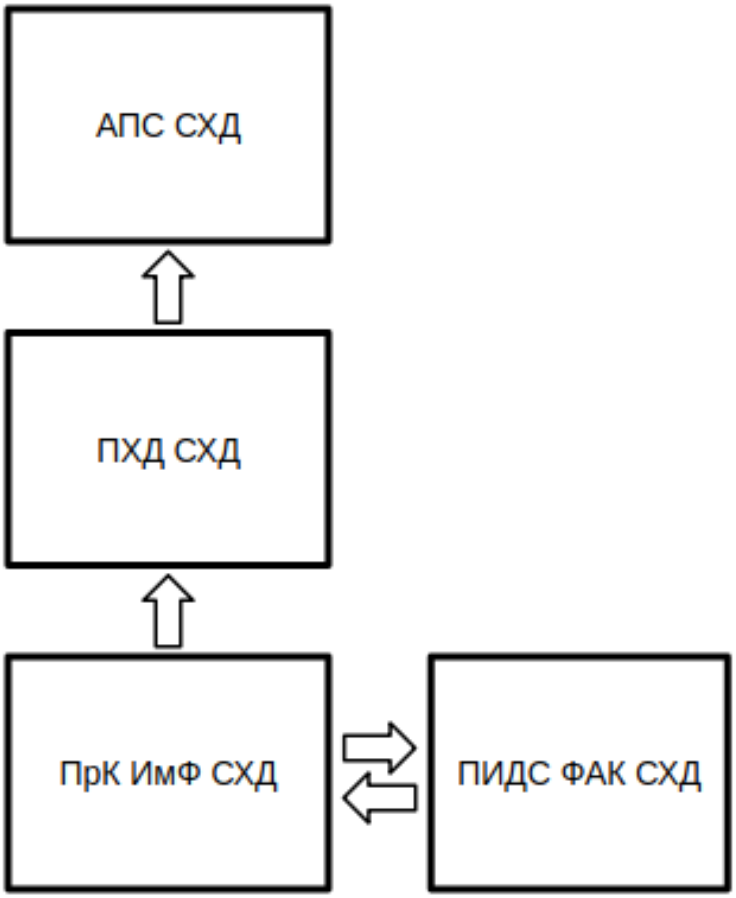

Рисунок 3 — Схема взаимодействия ПрК ИмФ с другими программами

1.7 ИСПОЛЬЗУЕМЫЕ ТЕХНИЧЕСКИЕ СРЕДСТВА

Реализация алгоритмов и прогоны симуляции требуют для себя программных ресурсов. В связи с тем, что реализация алгоритмов ПрК ИмФ проводится в рамках двух логических этапов, а именно:

- Дискретно-событийной модели
- Симуляции функционирования СХД

То для каждой из этих двух частей требуются разные необходимые ресурсы. Необходимые требования к аппаратному обеспечению для первого пункта приведены в таблице 5. Необходимые требования к аппаратному обеспечению приведены в таблице 6.

Элемент	Минимальные требования	Рекомендуемые
Процессор компьютера	1 ядра	2 ядер
Видео карта компьютера	–	–
Системная память	2 ГБ	4 ГБ
Дисковая подсистема	10 ГБ	20 ГБ

Таблица 5 — Аппаратные требования для реализации дискретно-событийной модели АПС СХД.

Элемент	Минимальные требования	Рекомендуемые
Процессор компьютера	2 ядра	4 ядра
Системная память	8 ГБ	16 ГБ
Дисковая подсистема	32 ГБ	64 ГБ

Таблица 6 — Аппаратные требования для прогона компьютерной реализации ПрК ИмФ.

1.8 ВЫЗОВ И ЗАГРУЗКА

Способ вызова программы с соответствующего носителя данных

Для работы ПрК ИмФ необходим компилятор языка программирования Go, не ниже версии 1.10. Для его установки нужно проделать следующие шаги:

– Скачать архив бинарников с официального сайта https://golang.org/dl/

– Распаковать его при помощи команды в терминале `tar -C /usr/local -xzf go`

– Установить переменную среды `export PATH=$PATH:/usr/local/go/bin`

– Запустить симуляцию `go run main.go platform.xml deployment.xml`, где `platform.xml` – путь к файлу, который содержит информацию о топологии системы, а `deployment.xml` – о начальных

1.9 ВХОДНЫЕ ДАННЫЕ

Входными данными ПрК ИмФ СХД должны являться:

– параметры имитации функционирования СХД;

– входные данные АД СХД и АИмФ СХД;

– параметры нагрузки на СХД;
– массив управляющих воздействий на компоненты СХД.

Характер, организация и предварительная подготовка входных данных

Формат, описание и способ кодирования входных данных

Входные данные приходят на вход в виде таблицы для каждого компонента СХД и СХД в целом.

1.10 ВЫХОДНЫЕ ДАННЫЕ

Программная реализация ПрК ИмФ СХД каждый дискретный промежуток времени сохраняет данные результатов работы алгоритма имитации функционирования СХД, что является одним из преимуществ симулятора. Данные содержат параметры компонент СХД:

– параметры контроллеров хранения:
  трафик контроллера хранения: в диапазоне от 0 до 5 ГБ/с;
  загрузка контроллера хранения: в диапазоне от 0 до 100 %;
– параметры носителей информации (НИ):
  тип (HDD, SSD);
  интерфейс передачи данных – PCI Express;
  емкость: в диапазоне от 0 до 12000 ГБ;
  свободное место на НИ: в диапазоне от 0 до 12000 ГБ;
  занятое место на НИ: в диапазоне от 0 до 12000 ГБ;
  средняя скорость чтения/записи: в диапазоне от 0 до 0,16 ГБ/с;
– параметры фабрики PCI Express:
  количество подключенных устройств: в диапазоне от 12 до 2304 шт.;
  максимальная пропускная способность: в диапазоне от 0,25 ГБ/с до 15 ГБ/с;
– параметры коммутатора внешней управляющей сети СХД:
  количество подключенных устройств: в диапазоне от 0 до 5 шт.;
  количество одновременно включенных коммутаторов: в диапазоне от 1 до 2 шт.;

– параметры функционирования СПО СХД:

объем передаваемых данных в режиме чтения/записи от 0 до 20 ТБ;

количество запросов на чтение/запись в единицу времени: в диапазоне от 0 до 1 000 шт/с;

время отклика на запросы чтения/записи: в диапазоне от 10 мс до 10 000 мс;

время обработки запросов на чтение/запись: в диапазоне от 10 мс до 10 000 мс;

способ обработки операций ввода-вывода: синхронный, асинхронный;

количество аннулированных запросов на чтение/запись в единицу времени: в диапазоне от 0 до 1000 шт/с.

– размер блока данных при чтении/записи: 4 КБ, 8 КБ, 16 КБ, 32 КБ, 64 КБ, 128 КБ, 256 КБ, 512 КБ, 1024 КБ, 1536 КБ, 2048 КБ;

– количество процессов, генерирующих запросы на чтение/запись, шт.: 1, 2, 4, 8, 16, 32, 64, 128, 160, 192;

– длина очереди запросов на чтение/запись для асинхронных операций ввода-вывода, шт.: 32, 64;

– состояния компонентов, в соответствии с ГОСТ 27.002–2015:

работоспособное состояние;

предотказное состояние (сбой);

неработоспособное состояние (частичный отказ);

предельное состояние (полный отказ).

Характер и организация выходных данных

Программная реализация ПрК ИмФ возвращает на выходе JSON-объект, содержание переменные, отвечающие за состояние компонент СХД и всего СХД. К наблюдаемым компонентам относятся:

– Контроллеры хранения, 4 шт.

– Носители информации, 12-2304 шт.

– Фабрика PCI Express, 1 шт.

– Коммутатора внешней управляющей сети СХД, 1 шт.

Формат, описание и способ кодирования выходных данных

Выходные данные представляются для каждого компонента СХД и СХД в целом (включая характеристики среды) в виде JSON-файла. Выбор данного формата был обусловлен простотой передачи объектов такого типа по сети. Элементы полей json-файла содержат численные значения наблюдаемых величин в заданный промежуток времени.

output.json

```
{
  "timestamp": "",
  "ambience": {
    "air_temp": 72,
    "humidity": 72,
    "atm_pressure": 90,
    "vibration": 32
  },
  "storage_components": [
    {
      "type": "controller",
      "id": "Anomaly",
      "name": "Anomaly",
      "props": {
        "status": 0,
        "dev_temp": 22,
        "uptime": 0.1,
        "traffic": 0,
        "load": 0
      }
    },
    {
      "type": "controller",
      "id": "SSD",
      "name": "SSD",
      "props": {
        "status": 0,
        "dev_temp": 22,
        "uptime": 0.1,
        "traffic": 0,
        "load": 0
      }
    },
    {
      "type": "controller",
      "id": "JBOD1",
```

```
      "name": "JBOD1",
      "props": {
        "status": 0,
        "dev_temp": 22,
        "uptime": 0.1,
        "traffic": 0,
        "load": 0
      }
    },
    {
      "type": "network_switch",
      "id": "NetworkSwitch",
      "name": "NetworkSwitch",
      "props": {
        "status": 0,
        "dev_temp": 22,
        "uptime": 0.1,
        "connected_dev_cnt": 3,
        "online_switches_cnt": 2
      }
    },
    {
      "type": "pci-fabric",
      "id": "FabricManager",
      "name": "FabricManager",
      "props": {
        "status": 0,
        "dev_temp": 22,
        "uptime": 0.1,
        "pcie_devices_cnt": 56,
        "pcie_max_bandw": 3
      }
    },
    {
      "type": "controller",
      "id": "Server1",
      "name": "Server1",
      "props": {
        "status": 0,
        "dev_temp": 22,
        "uptime": 0.1,
        "traffic": 29.749000000000425,
        "load": 0
      }
    },
    {
```

```
      "type": "controller",
      "id": "Server2",
      "name": "Server2",
      "props": {
        "status": 0,
        "dev_temp": 22,
        "uptime": 0.1,
        "traffic": 18.211999999999993,
        "load": 0
      }
    },
    {
      "type": "controller",
      "id": "Server3",
      "name": "Server3",
      "props": {
        "status": 0,
        "dev_temp": 22,
        "uptime": 0.1,
        "traffic": 9.833000000000007,
        "load": 0
      }
    },
    {
      "type": "controller",
      "id": "Server4",
      "name": "Server4",
      "props": {
        "status": 0,
        "dev_temp": 22,
        "uptime": 0.1,
        "traffic": 21.447000000000116,
        "load": 0
      }
    },
    {
      "type": "controller",
      "id": "JBOD2",
      "name": "JBOD2",
      "props": {
        "status": 0,
        "dev_temp": 22,
        "uptime": 0.1,
        "traffic": 0,
        "load": 0
      }
```

```
    },
    {
      "type": "io-balancer",
      "id": "LoadBalancer",
      "name": "LoadBalancer",
      "props": {
        "status": 0,
        "dev_temp": 22,
        "uptime": 0.1,
        "read_requests_rate": -0,
        "write_requests_rate": -0,
        "read_response_delay": 0,
        "write_response_delay": 1.9565000000000003,
        "read_data_volume": 0.1,
        "write_data_volume": 0.2,
        "read_request_process_time": 20,
        "write_request_process_time": 25,
        "io_processing_method": 0,
        "read_cancel_rate": 200,
        "write_cancel_rate": 300,
        "block_size": 64,
        "io_process_cnt": 8,
        "read_queue_length": 32,
        "write_queue_length": 64
      }
    },
    {
      "type": "controller",
      "id": "Client",
      "name": "Client",
      "props": {
        "status": 0,
        "dev_temp": 22,
        "uptime": 0.1,
        "traffic": 39.78299999999998,
        "load": 0
      }
    }
  ]
}
```

ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

Номера листов (страниц)					Всего листов (страниц) в докум	№ документа	Входящий № сопроводительного документа и дата	Подп.	Дата
Изм	измененных	замененных	новых	аннулированных					