Отчёт

ВШЭ

11 апреля 2018 г.

Содержание

1 Имитация СХД

Разработка СХД ведется в два этапа:

1. Разработка библиотеки, позволяющей создавать дискретно-событийные модели
2. Реализация модели, имитирующей СХД Татлин.

В главах 2 и 3 даётся описание каждой из этих частей, соответственно.

2 Реализация дискретно-событийной библиотеки Simlibrary для моделирования СХД

2.1 Особенности языка Go

Go (часто также Golang) — компилируемый многопоточный язык программирования. Он разрабатывался как язык программирования для создания высокоэффективных программ, работающих на современных распределённых системах и многоядерных процессорах. Среди его особенностей можно отметить:

- Простой и понятный синтаксис.
- Статическая типизация. Позволяет избежать ошибок, допущенных по невнимательности, упрощает чтение и понимание кода, делает код однозначным.
- Скорость и компиляция. Скорость у Go в десятки раз быстрее, чем у скриптовых языков, при меньшем потреблении памяти. При этом, компиляция практически мгновенна. Весь проект компилируется в один бинарный файл, без зависимостей. Как говорится, «просто добавь воды». И вам не надо заботиться о памяти, есть сборщик мусора.
- Отход от ООП. В языке нет классов, но есть структуры данных с методами. Наследование заменяется механизмом встраивания. Существуют интерфейсы, которые не нужно явно имплементировать, а лишь достаточно реализовать методы интерфейса.
- Параллелизм. Параллельные вычисления в языке делаются просто, изящно и без головной боли. Горутины (что-то типа потоков) легковесны, потребляют мало памяти.
- Богатая стандартная библиотека. В языке есть все необходимое для веб-разработки и не только. Количество сторонних библиотек постоянно растет. Кроме того, есть возможность использовать библиотеки C и C++.
- Возможность писать в функциональном стиле. В языке есть замыкания (closures) и анонимные функции. Функции являются объектами первого порядка, их можно передавать в качестве аргументов и использовать в качестве типов данных.

2.1.1 Типы данных в Go

В Go существуют следующие типы целых чисел: uint8, uint16, uint32, uint64, int8, int16, int32 и int64. 8, 16, 32 и 64 говорит нам, сколько бит использует каждый тип. uint означает «unsigned integer» (беззнаковое целое), в то время как int означает «signed integer» (знаковое целое). Беззнаковое целое может принимать только положительные значения (или ноль). В дополнение к этому существуют два типа-псевдонима: byte (то же самое, что uint8) и rune (то же самое, что int32).

Строка (string) — это последовательность символов определенной длины, используемая для представления текста. Строки в Go состоят из независимых байтов, обычно по одному на каждый символ.

В Go есть два вещественных типа: float32 и float64 (соответственно, часто называемые вещественными числами с одинарной и двойной точностью).

2.1.2 Как задаются функции в Go

Функция является независимой частью кода, связывающей один или несколько входных параметров с одним или несколькими выходными параметрами. Функции (также известные как процедуры и подпрограммы) можно представить в виде схемы, как показано на рисунке 1.

Рис. 1: Схематическое представление функции

В общем виде синтаксис функций представляется в виде (рис. **??**):

```
func average(xs []float64) float64
```

Рис. 2: Заголовок функции, где average представляеи имя функции, xs []float64 – список чисел типа float64, за скобками указывается тип возращаемого значения.

2.2 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). Функции 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 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.

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 данному хосту.

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

2.2.1 Функции Environment

func NewEnvironment() *Environment

Входные аргументы: отсутсвуют.

Выходное значение: переменнная типа *Environment.

Описание функции:Создаёт и инициализует необходимые поля для функцирования симуляции, такие как:

- queue
- workers

Таблица 1: Единицы системы измерений при моделировании

TB	1000^4 byte
GB	1000^3
MB	1000^2
KB	1000
B	1
GBps	1000^3 byte per sec
MBps	1000^2 byte per sec
KBps	1000 byte per sec
Bps	1 byte per sec
Gf	1000^3 flops
Mf	1000^2 flops
Kf	1000 flops
f	1 flops

– SendEventsNameMap

– ReceiveEventsNameMap

– ReceiverSendersMap

– stepEnd

– 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).
func (env *Environment) stopSimulation(EventInterface)
Входные аргументы: указатель на объект типа *Environment.
Выходное значение: отсутствует.
Описание: Даннная функция останавливает исполнение программы путем выставления флага shouldStop в положительное значение.
func (env *Environment) updateQueue(deltaTime float64)
Входные аргументы: указатель на объект типа *Environment.
Выходное значение: отсутствует.
Описание: Даннная функция обновляет очередь событий за время $deltaTime$.
func (env *Environment) CreateTransferEvents()
Входные аргументы: указатель на объект типа *Environment.
Выходное значение: отсутствует.
Описание: Даннная 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.

1. Создать события, которые имитируют передачу данных от одного хоста к другому.
2. Проверить является ли этот шаг симуляции последним.
3. Проверить симуляцию на возникновение дедлоков.
4. Получить событие из очереди с минимальным значением времени.
5. Обновить текущее время.
6. Обновить очередь событий за время, прошедшее с времени прошлого события.
7. Обработать коллбэки (callbaks) текущего события.
8. 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: Данная 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: Данная 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: Данная функция дожидается выполнения задач текущими горутинами, которым были посланы сигналы на предыдущем этапе.

2.3 Описание примитивов, использованных при имитации системы

2.3.1 Имитация сети

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

2.3.2 Имитация хоста

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.

2.3.3 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.

2.3.4 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.

2.3.5 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.

2.3.6 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.

2.3.7 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 .

func (p *Process) GetEnv() *Environment

Входные аргументы: Указатель на структуру, которая симулирует процесс.

Выходное значение: .

Описание функции: Данная функция .

2.3.8 Имитация процессов

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.

2.3.9 Функции

func (worker *Process) SendTask(task *Task, address string) STATUS return worker.SendTaskWithTimeout(task, address, -1)

func (worker *Process) SendTaskWithTimeout(task *Task, address string, timeout float64) STATUS event := NewSendEvent(worker, task, address) event.callbacks = append(event.callbacks, deleteSelfFromResource, event.deleteTimeOutEvents)

if timeout != -1 timeoutEvent := TimeOutEvent ConstantEvent: ConstantEvent Event: Event timeEnd: env.currentTime + timeout, worker: worker, timeout: timeout, , , halfEvent: event, event.timeOutEvent = timeoutEvent timeoutEvent.callbacks = append(timeoutEvent.callbacks, event.deleteSendEventFromGlobalQueue, event.deleteSendEventFromSend env.mutex.Lock() heap.Push(env.queue, timeoutEvent) env.mutex.Unlock()

env.mutex.Lock() env.SendEventsNameMap[address] = append(env.SendEventsNameMap[address], event) env.mutex.Unlock()

//should wait for an own turn env.stepEnd <- struct return <-worker.resumeChan

func (worker *Process) DetachedSendTask(task *Task, address string) interface

event := NewSendEvent(worker, task, address) event.async = true

event.callbacks = append(event.callbacks, deleteSelfFromResource)

env.mutex.Lock() env.SendEventsNameMap[address] = append(env.SendEventsNameMap[address], event) env.mutex.Unlock() return nil

func (worker *Process) ReceiveTask(address string) (*Task, STATUS) return worker.ReceiveTaskWithTimeout(address, -1)

func (worker *Process) ReceiveTaskWithTimeout(address string, timeout float64) (*Task, STATUS) event := ReceiveMock worker: worker, address: address,
if timeout != -1 timeOutEvent := TimeOutEvent ConstantEvent: ConstantEvent Event: Event timeEnd: env.currentTime + timeout, worker: worker, timeout: timeout, , , halfEvent: event, event.timeOutEvent = timeOutEvent timeOutEvent.callbacks = append(timeOutEvent.callbacks, event.deleteRecMockFromGlobalQueue, event.deleteRecFromReceiveEve env.mutex.Lock() heap.Push(env.queue, timeOutEvent) env.mutex.Unlock()
env.mutex.Lock()
if $_, ok := env.ReceiveEventsNameMap[address]; okpanic(fmt.Sprintf("multiplelistenonthesameaddre eventenv.mutex.Unlock()env.stepEnd < −structreturnevent.task, < −worker.resumeChan$
func (worker *Process) $\mathrm{SIM}_wait(waitTimefloat64)interface$
event := ConstantEvent Event timeStart: env.currentTime, timeEnd: env.currentTime + waitTime, worker: worker, , env.mutex.Lock() heap.Push(env.queue, event) env.mutex.Unlock()
// Wait for an own turn env.stepEnd <- struct <-worker.resumeChan return nil
func (worker *Process) WriteAsync(storage *Storage, task *Task) interface event := NewSendEvent(worker, task,) event.storage = true event.async = true event.link = storage.writeLink atomic.AddInt64(storage.writeLink.$_counter, 1$)
t := TransferEvent sendEvent: event, receiveEvent: nil, event.callbacks = append(event.callbacks, deleteSelfFromResource)
storage.writeLink.Put(t) return nil
func (worker *Process) Write(storage *Storage, task *Task) interface event := NewSendEvent(worker, task,) event.storage = true event.link = storage.writeLink atomic.AddInt64(storage.writeLink.$_counter, 1$)
t := TransferEvent sendEvent: event, receiveEvent: nil, event.callbacks = append(event.callbacks, deleteSelfFromResource)
storage.writeLink.Put(t)
env.stepEnd <- struct <-worker.resumeChan return nil
func (worker *Process) ReadAsync(storage *Storage, task *Task) interface event := NewSendEvent(worker, task,) atomic.AddInt64(storage.readLink.$_counter, 1$)
t := TransferEvent sendEvent: event, receiveEvent: nil,
event.callbacks = append(event.callbacks, deleteSelfFromResource)
storage.readLink.Put(t) return nil
func (worker *Process) Read(storage *Storage, task *Task) interface event := NewSendEvent(worker, task,)
atomic.AddInt64(storage.readLink.$_counter, 1$)
t := TransferEvent sendEvent: event, receiveEvent: nil, event.callbacks = append(event.callbacks, deleteSelfFromResource)
storage.readLink.Put(t)
// Wait for an own turn env.stepEnd <- struct <-worker.resumeChan return nil

2.4 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.

2.5 Имитация аномалий

В текущей версии реализации дискретно-событийной модели имплементировано два вида аномалий: хостов и сетей.

2.5.1 Написать, почему рассмотрено именно два вида аномалий. Если какие-то пропущены, то указать причину. Написать, почему именно эти виды аномалий

2.5.2 Аномалия сети

Аномалия сети моделируется при помощи следующей последовательности действий:

1. Выставить пропускную способность, трафик через данную сеть равными нулю.
2. Найти все пакеты, передаваемые по этим сетям.
3. Удалить пакеты из глобальной очереди событий.
4. Уведомить процессы, которые принимали/передавали пакеты об статусом передачи $FAIL$.

Реализация этой парадигмы показана на рис. 2.

Рис. 3: Аномалия сети

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 элемент из очереди событий.
func(eq *eventQueue) Fix(h Interface, i int)	h - элемент в очереди, который нуждается в изменениях. i - новый приоритет	-	Данная функция меняет приоритет у элемента h в очереди событий на приоритет i.

2.6 Написать какие примитивы используются для представленной последовательности действий аномалий сети

2.6.1 Аномалия хоста

Аномалия хоста моделируется при помощи следующей последовательности действий:

1. Выставить вычислительную мощность хоста равной нулю.
2. Найти все исходящие/входящие провода из/в данный хост.
3. Выставить пропусные способности (степень деградации), трафик через данные сети равными нулю.
4. Найти все пакеты, передаваемые по этим сетям.
5. Удалить их из глобальной очереди событий.
6. Уведомить процессы, которые принимали/передавали пакеты об статусом передачи $FAIL$.
7. Прекратить выполнение задач, исполняемых на данном хосте, со статусом $FAIL$.

Реализация этой парадигмы показана на рисунке 3.

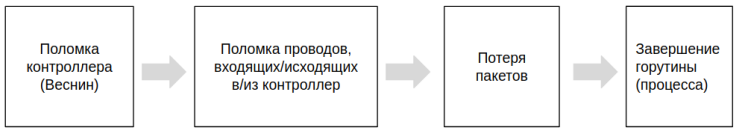

Рис. 4: Аномалия хоста

Восстановление хоста моделируется как показано на рис. 4 при следующей последовательности действий:

1. Выставить степень деградации данного хоста равной нулю
2. Найти все исходящие/входящие провода из/в данный хост.
3. Выставить степень деградации данных сетей равной нулю.

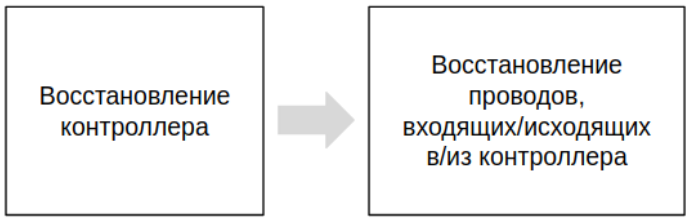

Рис. 5: Восстановление хоста

2.7 Написать какие примитивы используются для представленной последовательности действий аномалий хоста/восстановления хоста

3 Моделирование Татлина

3.0.1 Привести схему питерского стенда и дать текстовое описание КАЖДОГО ее компонента

3.0.2 Дать объяснение, как преобразовалась схема стенда в нашу с 4мя блоками (как они укрупнились)

3.0.3 timeout fail ok descrption)

3.1 Введение

Для моделирования СХД Татлин были реализованы функции, как показано на рис. 5:

- ClientExecutor (Клиент)
- LoadBalancer (БН)
- ServerExecutor (Контроллер (Веснин))
- PCIeFabric (PCIe фабрика)
  - CacheExecutorRead
  - CacheExecutorWrite
- DiskExecutor (Диск)

Балансировщик нагрузки (БН) при обращении клиента создаёт три соответсвующих процесса: один процесс отвечающий за логику серверной части, второй – за PCIe фабрику и третий – за конечный дисковый носитель. Описание каждого из них дано в последующих главах.

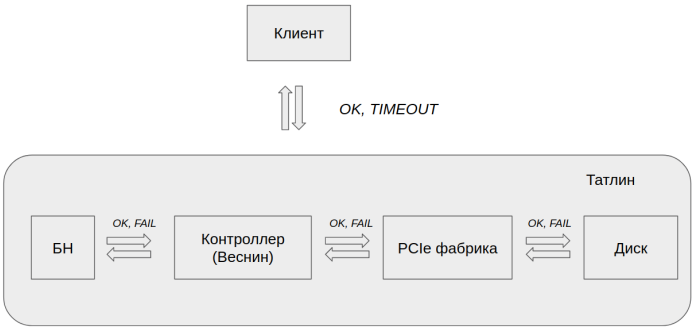

Рис. 6: Нормальный режим работы СХД Татлин. БН – балансировщик нагрузки

3.2 Описание Клиента

Клиентская часть реализует следующую последовательность действий:

1. Обращение к БН для того, чтобы узнать имя сервера, который будет обслуживать действующий запрос.
2. Инициализировать соединение с сервером
3. Цикл до тех пор пока не отправятся все пакеты:
   - Отправка пакета данных
     - В случае аномалии вернуться к пункту 1
   - Ожидание сообщения об успешной записи пакета на диск
     - В случае аномалии вернуться к пункту 1
4. Завершение соединения

3.3 Описание Балансировщика Нагрузки

Данный процесс завершит свою деятельность после завершения других процессов. Балансировщик Нагрузки реализует следующую последовательность действий:

- Бесконечный цикл:
  - Ожидание запроса на соединение от клиента
  - Установление соединения
  - Выбор контроллера, который будет обслуживать запросы от клиента
  - Завершение соединения

3.4 Описание Серверной части

Сервер реализует следующую последовательность действий:

- Установка соединения с клиентом
- Бесконечный цикл:
  - Ожидание пакета от клиента
    - Если произошла аномалия, то выйти из цикла
  - Отправка пакета PCIe фабрике
    - Если произошла аномалия, то выйти из цикла
  - Отправка клиенту сообщения об успешной записи пакета на диск
    - Если произошла аномалия, то выйти из цикла
    - Если пакет последний, то выйти из цикла

3.5 Описание PCIe фабрики

Фабрика реализует следующую последовательность действий:

- Установка соединения с сервером
- Бесконечный цикл:
  - Ожидание пакета от сервера
    - Если произошла аномалия, то выйти из цикла
  - Отправка пакета на запись в кэш
  - Отправка пакета на запись на диск
  - Отправка серверу сообщения об успешной записи пакета на диск
    - Если произошла аномалия, то выйти из цикла
    - Если пакет последний, то выйти из цикла

3.6 Описание Конечного дискового носителя

Конечный дисковый носитель реализует следующую последовательность действий.

- Установка соединения с PCIe фабрикой
- Бесконечный цикл:
  - Ожидание пакета от сервера
    - Если произошла аномалия, то выйти из цикла
  - Отправка пакета на запись в кэш
  - Отправка пакета на запись на диск
  - Отправка серверу сообщения об успешной записи пакета на диск
    - Если произошла аномалия, то выйти из цикла
    - Если пакет последний, то выйти из цикла

3.7 Аномалии в СХД Татлин

На данный в дискретно-событийной библиотеке реализовано два вида аномалий: серверов и сетей, что и нашло отражение при симулировании СХД Татлин.

Сценарий развития аномалии:

- Полностью выходит из строя один из серверов
- По получении сообщения о выхода из строя сервера, либо по таймауту по цепочке выходят из строя следующие процессы, обслуживающие передачу конкретного файла:
  - Клиент
  - Балансировщик нагрузки
  - PCIe фабрика
  - Диск

Схематическое представление выхода из строя сервера показано на рис. 6.

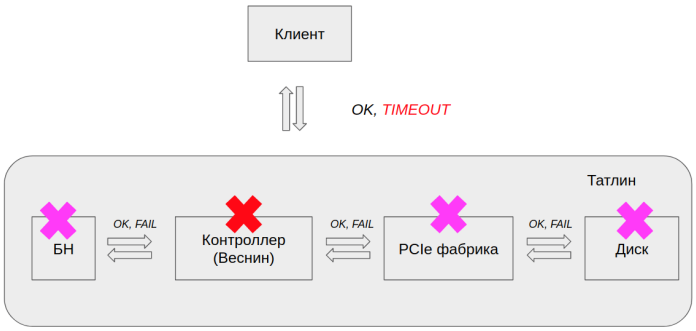

Рис. 7: Возникновение аномалии в СХД Татлин

3.8 Параметры запуска тестового прогона

Начальные параметры компонент СХД Татлин заданы в файле $platform.xml$. Параметры дисковых контроллеров, сетей, конечных дисковых носителей и отображены в таблицаъъ5, соответственно:

Таблица 5: Параметры дисковых котроллеров

	Имя	Скорость, Гфс
Storage controller 1	Server 1	1
Storage controller 2	Server 2	1
Storage controller 3	Server 3	1
Storage controller 4	Server 4	1

Таблица 7: Параметры сетей

Имя сети	Скорость, ГБ/с	Задержка сети, мс
Client_Server1	10	0
Client_Server2	10	0
Client_Server3	10	0
Client_Server4	10	0
Server1_FabricManager	12	0
Server2_FabricManager	12	0
Server3_FabricManager	12	0
Server4_FabricManager	12	0
FabricManager_SSD	12	0
FabricManager_JBOD1	2	0
FabricManager_JBOD2	2	0