Вопрос 1. Протоколы и веб-сервисы в распределенных системах.

1. Протоколы
Используются для обмена данными между узлами системы.

- **Транспортные:**
  - **TCP** — надёжная передача, контроль порядка и целостности.
  - **UDP** — быстрая передача без гарантий доставки.
- **Прикладные:**
  - **HTTP/HTTPS** — основной протокол для веб-взаимодействия.
  - **WebSocket** — двусторонняя постоянная связь.
  - **gRPC** — высокопроизводительный RPC поверх HTTP/2.
  - **AMQP, MQTT, Kafka protocol** — обмен сообщениями.

2. Веб-сервисы
Способ организации взаимодействия между компонентами через сеть.

- **Типы:**
  - **REST** — архитектурный стиль, использует HTTP-методы (GET, POST, PUT, DELETE).
  - **SOAP** — строгий протокол на основе XML и WSDL.
  - **GraphQL** — запросы к API с гибким выбором данных.

3. Роль в распределённых системах

- Обеспечивают **коммуникацию и интеграцию** компонентов.
- Поддерживают **масштабируемость, отказоустойчивость и совместимость**.
- Позволяют строить **микросервисную архитектуру**.

Вопрос 2. Групповые взаимодействия и рассылка. Непрямое взаимодействие в распределенных системах.

1. Групповые взаимодействия
Связь типа **«один-ко-многим»** или **многие-ко-многим**.

- **Мультикаст** — сообщение доставляется группе узлов.
- **Бродкаст (Broadcast)** — сообщение отправляется всем участникам сети.
- **Anycast** — сообщение доставляется одному из доступных узлов группы.

2. Рассылка (Message dissemination)
Механизмы распределения сообщений:

- **Push** — отправитель инициирует доставку.
- **Pull** — получатели сами запрашивают сообщения.
- **Gossip-протоколы** — случайное распространение сообщений по узлам (эпидемическая модель).

3. Непрямое взаимодействие
Узлы **не обмениваются сообщениями напрямую**, а используют посредника.

Основные модели:

- **Publish/Subscribe (Pub/Sub)** — издатели публикуют события, подписчики получают их по темам.
- **Очереди сообщений (Message Queues)** — асинхронная передача через брокера.
- **Событийно-ориентированная модель (Event-driven)** — реакции на события без прямых вызовов.

4. Преимущества непрямого взаимодействия

- Слабая связность компонентов.
- Масштабируемость.
- Повышенная отказоустойчивость.

Вопрос 3. Обнаружение отказов. Масштабирование.

1. Обнаружение отказов (Failure Detection)
Механизмы выявления неработающих узлов.

- **Таймауты** — если узел не отвечает за заданное время, считается отказавшим.
- **Heartbeat-сообщения** — периодические сигналы «я жив».
- **Пассивный и активный мониторинг** — через наблюдающие узлы или центральные координаторы.

Типы ошибок:

- **Crash-failure** — узел полностью перестаёт работать.
- **Omission-failure** — потеря сообщений.
- **Byzantine-failure** — произвольное (вредоносное/непредсказуемое) поведение.

2. Масштабирование (Scalability)
Способность системы сохранять производительность при росте нагрузки.

Виды масштабирования:

- **Вертикальное (Scale Up)** — увеличение ресурсов одного узла (CPU, RAM).
- **Горизонтальное (Scale Out)** — добавление новых узлов в систему.

Основные механизмы:

- **Балансировка нагрузки (Load Balancing)**.
- **Шардирование данных (Sharding)**.
- **Репликация (Replication)**.
- **Кэширование (Caching)**.

Цели:

- Высокая доступность.
- Устойчивость к отказам.
- Линейный рост производительности.

Вопрос 4. Параллельная обработка данных в распределенных системах

1. Суть

Одновременное выполнение вычислений на **нескольких узлах** для ускорения обработки больших объёмов данных.

2. Основные модели

- **Data Parallelism (параллелизм по данным)** — разбиение набора данных на части и обработка их параллельно.
- **Task Parallelism (параллелизм по задачам)** — разные задачи выполняются одновременно.
- **Pipeline-параллелизм** — пошаговая обработка по конвейеру.

3. Технологии и подходы

- **MapReduce** — разбиение на этапы *Map* и *Reduce*.
- **Spark** — распределённые вычисления в памяти.
- **Actor-модель** — независимые акторы обмениваются сообщениями.

4. Преимущества

- Ускорение вычислений.
- Масштабируемость.
- Эффективная обработка Big Data.

5. Проблемы

- Синхронизация.
- Балансировка нагрузки.
- Межузловые задержки.

ПРАКТИКА РАЗБОР

1. Реализовать запросы к удаленному серверу по протоколу HTTP с использованием утилит **telnet**, **curl** к следующим ресурсам:

http://www.mgpu.ru/

https://bmstu.ru/

https://cbr.ru/

Установка необходимого:

```
sudo apt update
sudo apt install -y curl telnet openssl
```

Если возвращает ошибку

E: Could not get lock /var/lib/apt/lists/lock. It is held by process 1579 (packagekitd) и тд:

Команды:

```
sudo kill -9 1579
sudo rm -f /var/lib/apt/lists/lock
sudo rm -f /var/cache/apt/archives/lock
sudo rm -f /var/lib/dpkg/lock*
sudo dpkg --configure -a
sudo apt update
```

Использование curl для доступа к www.mgpu.ru:

```
curl -i http://www.mgpu.ru/
```

```
mgpu@mgpu-vm:~$ curl -i http://www.mgpu.ru/
HTTP/1.1 301 Moved Permanently
Server: ddos-guard
Date: Sun, 07 Dec 2025 20:12:36 GMT
Connection: keep-alive
Keep-Alive: timeout=60
Set-Cookie: __ddg8_=ZtAdFmze8HFuXYYk; Domain=.mgpu.ru; Path=/; Expires=Sun, 07-D
ec-2025 20:32:36 GMT
Set-Cookie: __ddg10_=1765138356; Domain=.mgpu.ru; Path=/; Expires=Sun, 07-Dec-20
25 20:32:36 GMT
Set-Cookie: __ddg9_=178.217.100.86; Domain=.mgpu.ru; Path=/; Expires=Sun, 07-Dec
-2025 20:32:36 GMT
Location: https://www.mgpu.ru/
Content-Type: text/html; charset=utf-8
Content-Length: 568

<!DOCTYPE html><html lang=en><meta charset=utf-8><meta name=viewport content="in
itial-scale=1, minimum-scale=1, width=device-width"><title>Error 301</title><sty
le>*{margin:0;padding:0}html{font:15px/22px arial,sans-serif;background: #fff;co
lor:#222;padding:15px}body{margin:7% auto 0;max-width:390px;min-height:180px;pad
```

!!! `telnet` подходит только для старых HTTP-сайтов без редиректов/защиты. Современные сайты нужно запрашивать через **curl** или **openssl**. Поэтому для www.mgpu.ru использовали **curl.**

Доп. опции для: **`curl -i http://www.mgpu.ru/`**

- `-i` — показать заголовки + тело.
- `-I` — только заголовки (HEAD).
- `-L` — следовать перенаправлениям.
- `-v` — подробный вывод (sockets/handshake).

Перенаправление **301 Moved Permanently** означает, что сервер сообщает:

- **Этот URL больше не актуален**.
- **Ресурс перемещён навсегда** на другой адрес, указанный в заголовке `Location:`.
- Клиенты (браузеры, curl, поисковые системы) **должны использовать новый URL** в будущем.

Пример на практике:

- Ты заходишь на `http://www.mgpu.ru/`
- Сервер отвечает 301 с `Location: https://www.mgpu.ru/`
- Браузер или curl автоматически переходят на `https://www.mgpu.ru/`
- Теперь все последующие запросы лучше делать сразу на `https://www.mgpu.ru/`

То есть это **официальный способ сервера сообщить о постоянном переходе на новый адрес**, чаще всего для перевода с HTTP на HTTPS.

HTTPS (https://bmstu.ru/, https://cbr.ru/) — через curl

```
curl -i -k https://bmstu.ru/
```

```
mgpu@mgpu-vm:~$ curl -i -k https://bmstu.ru/
HTTP/1.1 301 Moved Permanently
Server: nginx
Date: Sun, 07 Dec 2025 20:35:05 GMT
Content-Type: text/html
Content-Length: 178
Connection: keep-alive
Location: https://mirror.bmstu.ru/
X-Robots-Tag: noindex, nofollow
Strict-Transport-Security: max-age=31536000
Content-Security-Policy: block-all-mixed-content
X-Frame-Options: DENY

<html>
<head><title>301 Moved Permanently</title></head>
<body bgcolor="white">
<center><h1>301 Moved Permanently</h1></center>
<hr><center>nginx</center>
</body>
</html>
```

!!! Сайт **bmstu.ru** использует **просроченный SSL-сертификат**. По умолчанию `curl` проверяет сертификаты и блокирует соединение, если они недействительны → ошибка `(60) SSL certificate problem`.

Флаг `-k` (`--insecure`) **отключает проверку подлинности сертификата**, позволяя получить страницу несмотря на просрочку.

То есть:

- Без `-k` → `curl` защищается, соединение не устанавливается.
- С `-k` → соединение устанавливается, сервер отвечает HTML, но проверка безопасности пропущена.

```
curl -i https://cbr.ru/
```

- **HTTP/2 200**
  - Сервер успешно обработал запрос.
  - Код **200 OK** — страница доступна.
- **server: ddos-guard**
  - Сайт использует защиту от DDoS (сервис `ddos-guard`).
- **set-cookie**
  - Сервер отправляет куки браузеру/клиенту.
  - Используются для сессий, идентификации пользователя или защиты от атак.

Альтернатива через OpenSSL (raw-запрос):

```
printf "GET / HTTP/1.1\r\nHost:
bmstu.ru\r\nConnection: close\r\n\r\n" | \
openssl s_client -connect bmstu.ru:443 -servername
bmstu.ru -verify 0
```

```
mgpu@mgpu-vm:~$ printf "GET / HTTP/1.1\r\nHost: bmstu.ru\r\nConnection: close\r\
n\r\n" | \
>
>   openssl s_client -connect bmstu.ru:443 -servername bmstu.ru 2>/dev/null
CONNECTED(00000003)
---
Certificate chain
 0 s:CN = bmstu.ru
   i:C = US, O = Let's Encrypt, CN = R10
 1 s:C = US, O = Let's Encrypt, CN = R10
   i:C = US, O = Internet Security Research Group, CN = ISRG Root X1
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIE7TCCA9WgAwIBAgISBmz+Vt1VeEdDKO/Ep6weksGYMA0GCSqGSIb3DQEBCwUA
```

```
printf "GET / HTTP/1.1\r\nHost: cbr.ru\r\nConnection:
close\r\n\r\n" | \
openssl s_client -connect cbr.ru:443 -servername
cbr.ru
```

```
mgpu@mgpu-vm:~$ printf "GET / HTTP/1.1\r\nHost: cbr.ru\r\nConnection: close\r\n\
r\n" | \
>
> openssl s_client -connect cbr.ru:443 -servername cbr.ru

CONNECTED(00000003)
depth=2 OU = GlobalSign Root CA - R3, O = GlobalSign, CN = GlobalSign
verify return:1
depth=1 C = BE, O = GlobalSign nv-sa, CN = GlobalSign GCC R3 DV TLS CA 2020
verify return:1
depth=0 CN = *.cbr.ru
verify return:1
---
Certificate chain
```

Итого:

Команда curl — основной способ, работает с HTTPS, редиректами и заголовками.

OpenSSL — для демонстрации «ручного» запроса через TLS (протокол защиты соединений в интернете).

Проще говоря:

- **OpenSSL** позволяет нам напрямую подключиться к серверу через **защищённое соединение HTTPS**.
- Обычно браузер сам делает HTTPS-запросы, но с OpenSSL мы делаем это **вручную**: сначала устанавливаем шифрованное соединение, потом сами отправляем HTTP-запрос.
- «Ручной» значит, что мы сами пишем запрос (GET / …), а не браузер или curl делают это автоматически.

То есть это способ **посмотреть, как сервер отвечает через HTTPS**, шаг за шагом, и увидеть сертификат, шифрование и сам ответ.

Telnet — не используется, так как сайты защищены HTTPS и DDoS-щитами.

2. Реализовать рассылку сообщений с помощью IP Multicast. На примере файлов socket_multicast_sender.py и socket_multicast_receiver.py реализовать рассылку сообщений.

Терминал №1 — получатель. Он будет «слушать» сообщения.

```
python3 socket_multicast_receiver.py
```

```
mgpu@mgpu-vm:~/Downloads$ python3 socket_multicast_receiver.py

waiting to receive message
received 19 bytes from ('192.168.14.78', 40608)
b'very important data'
sending acknowledgement to ('192.168.14.78', 40608)

waiting to receive message
received 19 bytes from ('192.168.14.78', 36070)
b'very important data'
sending acknowledgement to ('192.168.14.78', 36070)

waiting to receive message
```

- **waiting to receive message** — программа ждёт, когда придёт сообщение.
- **received 19 bytes … b'very important data'** — сообщение пришло, 19 байт, текст `very important data`.
- **sending acknowledgement** — отправляет подтверждение обратно отправителю, что сообщение получено.
- Потом снова ждёт новые сообщения.

Терминал №2 — отправитель. Он будет отправлять сообщения.

```
python3 socket_multicast_sender.py
```

```
mgpu@mgpu-vm:~/Downloads$ python3 socket_multicast_sender.py
sending b'very important data'
waiting to receive
received b'ack' from ('192.168.14.78', 10000)
waiting to receive
timed out, no more responses
closing socket
mgpu@mgpu-vm:~/Downloads$
```

- **sending …** — сообщение отправлено всем на multicast-адрес.
- **waiting to receive** — ждёт подтверждения (ack) от приёмника.
- **received b'ack'** — получил подтверждение, что приёмник получил сообщение.
- **timed out, no more responses** — больше нет новых подтверждений, завершает работу.
- **closing socket** — закрывает соединение.

Итог простыми словами

- Ты «разослал сообщение» по специальному сетевому адресу.
- Кто слушал этот адрес — получил сообщение.
- Приёмник «помахал рукой» (отправил ack), что всё дошло.
- После этого отправитель закончил работу.
- То есть рассылка через multicast с подтверждением работает правильно.

3. Реализовать механизмы безопасности в распределенной системе с использованием сертификатов X.509 для аутентификации. Создайте распределенную систему, состоящую из:

- Сервера распределенной системы.
- координатора распределенной системы.
- одного клиента распределенной системы HTTPS.
- одного клиента распределенной системы HTTP.

Материалы для подготовки размещены на ресурсе

https://github.com/BosenkoTM/Distributed_systems/tree/main/practice/2024/lw_05

Что нужно сделать по сути

Мы должны собрать **маленькую распределённую систему** с четырьмя компонентами:

1. **Server (server.py)** – основной сервер, который принимает запросы от клиентов.
2. **Coordinator (coordinator.py)** – «координатор», который управляет взаимодействием серверов/клиентов (например, распределяет задания или проверяет аутентификацию).
3. **Client HTTPS (client.py)** – клиент, который подключается к серверу через **защищённое соединение HTTPS с сертификатом X.509**.
4. **Client HTTP (client2.py)** – клиент, который подключается через обычный HTTP (не защищённый).

Для чего нужны X.509 сертификаты

- **Сертификаты X.509** — это «паспорт» для компьютеров в сети.
- Они позволяют серверу **проверить подлинность клиента**, а клиенту — убедиться, что он соединяется с настоящим сервером.
- Используется HTTPS, чтобы зашифровать данные и проверить, что никто не подменяет сервер

Действия:

Файлы **НЕ как в гите, изменены**, скачаны в загрузки, переходим

```
cd ~/Downloads
```

Проверка версии и установки

```
python3 --version
```

```
pip3 install cryptography requests flask
```

- `cryptography` — для работы с сертификатами.
- `requests` — для HTTPS клиента.
- `flask` — если сервер сделан на Flask (часто так в учебных проектах).

Далее:

```
# Генерация ключей и сертификатов сервера
openssl req -newkey rsa:2048 -nodes -keyout server_key.pem -x509 -days 365 -out server_cert.pem -subj "/C=RU/ST=Moscow/L=Moscow/O=MGPU/OU=CS/CN=localhost"

# Генерация ключей и сертификатов клиента
openssl req -newkey rsa:2048 -nodes -keyout client_key.pem -x509 -days 365 -out client_cert.pem -subj "/C=RU/ST=Moscow/L=Moscow/O=MGPU/OU=CS/CN=localhost"

# CA для сервера (самоподписанный)
cp client_cert.pem ca_cert.pem

# Генерация ключа для шифрования данных (Fernet)
python3 - <<EOF
from cryptography.fernet import Fernet
key = Fernet.generate_key()
with open("encryption_key.txt", "wb") as f:
    f.write(key)
EOF
```

Далее запуск всех компонентов в отдельных терминалах:

1. **Терминал 1** — сервер HTTPS:

```
python3 server.py
```

```
mgpu@mgpu-vm:~/Downloads$ python3 server.py
 * Serving Flask app 'server'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment.
 Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on https://127.0.0.1:5000
 * Running on https://192.168.14.78:5000
Press CTRL+C to quit
```

2. **Терминал 2** — координатор:

```
python3 coordinator.py
```

```
mgpu@mgpu-vm:~/Downloads$ python3 coordinator.py
 * Serving Flask app 'coordinator'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment.
 Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:8000
 * Running on http://192.168.14.78:8000
Press CTRL+C to quit
```

3. **Терминал 3** — клиент HTTPS:

```
python3 client.py
```

```
mgpu@mgpu-vm:~/Downloads$ python3 client.py
/usr/lib/python3/dist-packages/urllib3/connectionpool.py:1004: InsecureRequestWa
rning: Unverified HTTPS request is being made to host 'localhost'. Adding certif
icate verification is strongly advised. See: https://urllib3.readthedocs.io/en/l
atest/advanced-usage.html#ssl-warnings
  warnings.warn(
Ответ сервера: {'result': 'ok'}
```

4. **Терминал 4** — клиент HTTP:

```
python3 client2.py
```

```
mgpu@mgpu-vm:~/Downloads$ python3 client2.py
/usr/lib/python3/dist-packages/urllib3/connectionpool.py:1004: InsecureRequestWa
rning: Unverified HTTPS request is being made to host 'localhost'. Adding certif
icate verification is strongly advised. See: https://urllib3.readthedocs.io/en/l
atest/advanced-usage.html#ssl-warnings
  warnings.warn(
Ответ сервера: {'result': 'ok'}
```

Если все четыре компонента запустились и работают:

- **server.py** — HTTPS сервер расшифровывает данные и возвращает `{'result':'ok'}`.
- **client.py** — HTTPS клиент успешно отправляет данные и получает ответ.
- **client2.py** — второй клиент (HTTP/HTTPS) тоже успешно подключается и получает ответ.
- **coordinator.py** — при необходимости пересылает запросы на сервер и возвращает результат.

В терминале сервера видно, что данные **получены и расшифрованы**

```
Получено и расшифровано: some_data
127.0.0.1 - - [08/Dec/2025 01:05:07] "POST /api/data HTTP/1.1" 200 -
Получено и расшифровано: some_data
127.0.0.1 - - [08/Dec/2025 01:07:48] "POST /api/data HTTP/1.1" 200 -
```

4. Определить сходимость SWIM, интервалы конвергенции (указать процент конвергенции, при котором график имеет наибольшее отклонение от гладкой кривой) компьютерной распределенной сети при следующих входных параметрах:

GOSSIP FANOUT -10, 5, 3 nodes.

GOSSIP INTERVAL – 0.1

NODES = 50, 100

PACKET LOSS – 10%, 50%.

Материалы для подготовки размещены на ресурсе

http://95.131.149.21/moodle/mod/assign/view.php?id=935

Установки

```
sudo apt update
sudo apt install -y python3-pip
pip3 install numpy pandas matplotlib scipy tqdm
```

Создать папку и скрипты:

```
mkdir -p ~/Downloads/swim_results
cd ~/Downloads
# допустим, swim_sim.py — скрипт симуляции, run_experiments.sh — управляющий скрипт
```

Нужно создать файлы:

- **Скрипт симуляции `swim_sim.py` (готовый, запускаемый)** — я могу прислать полный код, который ты запускаешь в VM, и который сразу генерирует CSV.
- **Скрипт анализа `analyze.py`** — строит графики и вычисляет все метрики.
- **run_experiments.sh** — обернёт всё в батч и запустит по всем конфигурациям.

Файл сделать исполняемым:

```
chmod +x ~/Downloads/run_experiments.sh
```

Запуск одного эксперимента

```
# пример: nodes=50, fanout=5, loss=0.1
python3 swim_sim.py --nodes 50 --fanout 5 --loss 0.1 --interval 0.1 --runs 30 --max-time 120 --outdir ~/Downloads/swim_results/test1 --processes 4
```

После окончания будет CSV файл. Запустить анализ:

```
python3 analyze.py --csv ~/Downloads/swim_results/test1/swim_nodes50_fan5_loss10.csv --outdir ~/Downloads/swim_results/test1
```

В каталоге `test1` появится PNG с графиком и `*_metrics.txt` с метриками (включая `percent_at_max_dev` и `t10_90`).

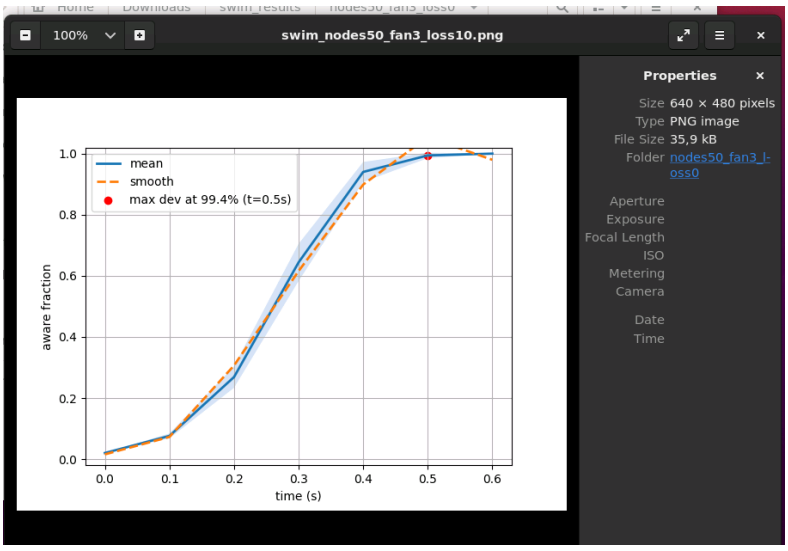
Запуск всех конфигураций

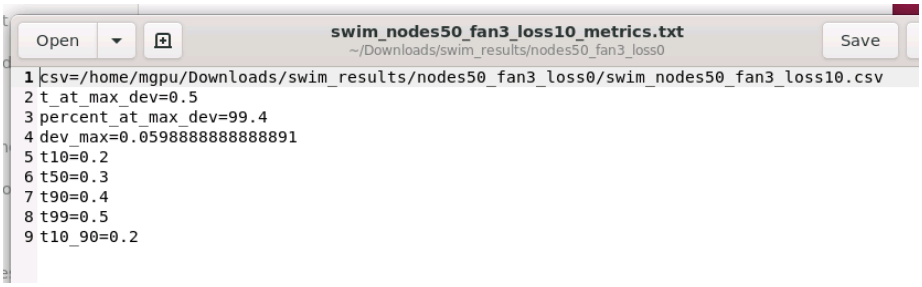
`~/Downloads/run_experiments.sh`

Это запустит 12 конфигураций по порядку (может занять время — зависит от runs и max_time). Результаты будут в `~/Downloads/swim_results/…` — по подпапкам.

```
mgpu@mgpu-vm:~/Downloads$ ~/Downloads/run_experiments.sh
Running nodes=50 fanout=10 loss=0.1 -> /home/mgpu/Downloads/swim_results/nodes50
_fan10_loss0
Saved: /home/mgpu/Downloads/swim_results/nodes50_fan10_loss0/swim_nodes50_fan10_
loss10.csv
Found CSV: /home/mgpu/Downloads/swim_results/nodes50_fan10_loss0/swim_nodes50_fa
n10_loss10.csv
✅ Analysis saved: /home/mgpu/Downloads/swim_results/nodes50_fan10_loss0/swim_no
des50_fan10_loss10.png /home/mgpu/Downloads/swim_results/nodes50_fan10_loss0/swi
m_nodes50_fan10_loss10_metrics.txt
Running nodes=50 fanout=10 loss=0.5 -> /home/mgpu/Downloads/swim_results/nodes50
_fan10_loss0
Saved: /home/mgpu/Downloads/swim_results/nodes50_fan10_loss0/swim_nodes50_fan10_
loss50.csv
Found CSV: /home/mgpu/Downloads/swim_results/nodes50_fan10_loss0/swim_nodes50_fa
n10_loss10.csv
✅ Analysis saved: /home/mgpu/Downloads/swim_results/nodes50_fan10_loss0/swim_no
```

В папках появятся графики и текстовые файлы, пример





Наши готовые папки

```
mgpu@mgpu-vm:~/Downloads$ cd ~/Downloads/swim_results
mgpu@mgpu-vm:~/Downloads/swim_results$ ls
nodes100_fan10_loss0  nodes100_fan5_loss0  nodes50_fan3_loss0
nodes100_fan3_loss0   nodes50_fan10_loss0  nodes50_fan5_loss0
mgpu@mgpu-vm:~/Downloads/swim_results$
```

Сжимаем файл и на ПК качаем

```
cd ~/Downloads
zip -r swim_results.zip swim_results
```

Далее анализ через колаб, всё в блокноте