

236370 תכנות מקבילי וUMBRELLA לעיבוד
נתונים ולמידת מכונה

דוח תרגיל בית 3

מリンה ינובסקי 324515659

קורן מעברי 207987314

```
# we want to collect data from all the processes.
# so, we will send from each process it's send_buffer to all the other processes.
# then, we will wait for all the data to be received.
# when all the data is in place, calculate reduction in each process into recv_buffer.

# allocate buffer for receiving values from all the process.
all_values = [np.empty_like(send) for _ in range(size)]

# send buffer to all other processes, and copy the send_buffer for the current process.
for i in range(size):
    if i != rank:
        req = comm.Isend(send, dest=i)

# receive back shared data from other processes
for i in range(size):
    if i != rank:
        req = comm.Irecv(all_values[i], i)
        # wait for all the values!
        req.Wait()

# reduce everyone
compute = send
for i in range(size):
    if i != rank:
        compute = np.vectorize(op)(all_values[i], compute)

recv[:] = compute
return recv

# first, we want to pass values to all the processes in ring and [op] them up.
# each iteration, a process sends a message (contains a chunk of the data) and receives one,
# and then it needs to perform the op on the values it got with its array.
# after performing n-1 [ops], we will need to distribute them.

# prepare for each process the data it will send in chunks
chunks = np.array_split(send, num_processes)

# prepare for each process buffer for receiving messages
recv_buffers = [np.empty_like(chunk) for chunk in chunks]

# the processes to communicate with
send_to = (rank + 1) % num_processes
recv_from = (rank - 1) % num_processes

# n-1 iterations for operation
for i in range(num_processes - 1):
    # prepare data to send
    send_idx = (rank - i) % num_processes
    msg = chunks[send_idx]
    req = comm.Isend(msg, send_to)

    # receive the data
    recv_idx = (rank - 1 - i) % num_processes # the index of the chunk to receive message and work with
    recv_req = comm.Irecv(recv_buffers[recv_idx], recv_from)
    recv_req.Wait()

    # perform operation
    if recv_buffers[recv_idx].size != 0:
        chunks[recv_idx] = op(chunks[recv_idx], recv_buffers[recv_idx])
        chunks[recv_idx] = np.vectorize(op)(chunks[recv_idx], recv_buffers[recv_idx])

# n-1 iterations for distribution
for i in range(num_processes - 1):
    # prepare data to send
    send_idx = (rank - i + 1) % num_processes
    msg = chunks[send_idx]
    req = comm.Isend(msg, send_to)

    # receive the data
    recv_idx = (rank - i) % num_processes # the index of the chunk to receive message and work with
    recv_req = comm.Irecv(recv_buffers[recv_idx], recv_from)
    recv_req.Wait()

    # save result
    if recv_buffers[recv_idx].size != 0:
        chunks[recv_idx] = recv_buffers[recv_idx]

# put result into recv buffer
recv[:] = np.concatenate(chunks, axis=0)
return recv
```

שאלה 1

:Naïve_allreduce

נקצה מקום למידע שיגיע מכל התהיליכים.

מכל תהיליך, נשלח את send לכל שאר התהיליכים.

התהיליך יוכל לקבל את כל המידע שכל שאר התהיליכים שלחו אליו (send) שנשלח משאר התהיליכים. לפwi שתהיליך ימשיך בחישוב reduction, עליו לחכות לקבלת המידע.

לאחר שכל המידע הגיע לתהיליך, הוא יבצע reduction עם הפעולה שקיבל בקלט.

לאחר מכן נעתיק את התוצאה לתוכה buffer שהוקצתה לה, לתוכו recv. בסיום הפעולה לכל תהיליך יהיה buffer ובו תוצאה החסוסה של כל המרכיבים מכל התהיליכים (בדומה לתוצאה reduction).

:Ring_allreduce

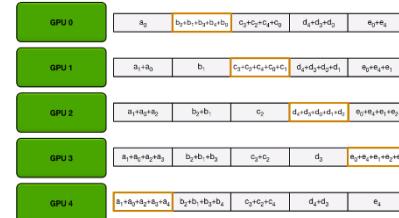
כל תהיליך כעת לא ישלח את כל הbuffer של המידע לשיליחה, אלא חלקים ממנו. נחלק את הbuffer לצ'אנקים כמספר התהיליכים, כי נרצה שעומס העבודה יתחלק בצורה כמה שיותר שוויונית בין כל התהיליכים.

נקצה גם כן buffer למידע שתהיליך יקבל בכל איטרציה.

נשים לב שתהיליך יוכל מידע מטהיליך מסוים וישלח מידע לתהיליך מסוים. נקרה להם מטהיליכים א'-ב' בהתאם. תהיליך א' הוא הקודם בסדר לתהיליך הנוכחי, וטהיליך ב' הוא הבא בסידור התהיליכים (באופן שריאנו בכיתה).

בהינתן ח-טהיליכים, נctrkr-1-ח איטרציות לביצוע הפעולות ב佗. מוגלאים (כפי שראינו בכיתה). נשלח בכל איטרציה באופן מעגלי לתהיליך א', צ'אנק, ונשלח צ'אנק שהגיע מטהיליך ב'. לאחר שנחכח לקלטתו, נבצע reduction לצ'אנק המתאים לו עם הפעולה בקלט.

בסוף האיטרציות, נראה את הbuffer של קבלת המידע כך:



cut, נctrkr-1-ח איטרציות להפצת צ'אנק המידע שמכל אחד התוצאה הסופית בין כל התהיליכים. הפצת זו תבוצע גם כן בצורה מוגלאה, כאשר כל תהיליך ישלח צ'אנק מעדכן לתהיליך א', ייחכה לקלט צ'אנק מעדכן מטהיליך ב', ולאחר מכן ישמר את המידע שקיבל במקומו המתאים לו.

בסוף כל האיטרציות, נעתיק לתוכו buffer ה recv את התוצאה של החשוב Buffer זה יראה כך בין כל התהיליכים:



```
# divide total number of batches between all the workers.
num_batches_per_worker = self.number_of_batches // self.num_workers
self.number_of_batches = num_batches_per_worker

# needed to set up num_of_batches in order to create minibatches properly!
# because num_of_batches minibatches are created when calling create_batches() method later in each epoch.

for epoch in range(self.epochs):
    # creating batches for epoch
    data = training_data[0]
    labels = training_data[1]
    mini_batches = self.create_batches(data, labels, self.mini_batch_size)
    for x, y in mini_batches:
        # do work - don't change this
        self.forward_prop(x)
        nabla_b, nabla_w = self.back_prop(y)

        # send nabla_b, nabla_w to masters
        # TODO: add your code

    for layer in range(self.num_layers):
        # we want to send the data to the master in charge of that layer.
        # we want to specify a tag indicating the layer number.
        # we will map the layers to tags as follows:
        # for biases: tag[layer] = 2 * layer, for weights: tag[layer] = 2 * layer + 1

        tag_base = 2 * layer
        master = layer % self.num_masters
        self.comm.Isend(nabla_w[layer], master, tag_base + 1)
        self.comm.Isend(nabla_b[layer], master, tag_base)

    # receive new self.weight and self.biases values from masters
    # TODO: add your code

    for layer in range(self.num_layers):
        tag_base = 2 * layer
        master = layer % self.num_masters

        req_w = self.comm.Irecv(self.weights[layer], master, tag_base + 1)
        req_w.Wait()

        req_b = self.comm.Irecv(self.biases[layer], master, tag_base)
        req_b.Wait()

# setting up the layers this master does
nabla_w = []
nabla_b = []
for i in range(self.rank, self.num_layers, self.num_masters):
    nabla_w.append(np.zeros_like(self.weights[i]))
    nabla_b.append(np.zeros_like(self.biases[i]))

self.number_of_batches = (self.number_of_batches // self.num_workers) * self.num_workers
# because we changed num_of_batches in workers- we need them to match each other

for epoch in range(self.epochs):
    for batch in range(self.number_of_batches):

        # wait for any worker to finish batch and
        # get the nabla_w, nabla_b for the master's layers
        # TODO: add your code

        # get the worker id to receive the message from
        status = MPI.Status()
        self.comm.Probe(source=MPI.ANY_SOURCE, tag=MPI.ANY_TAG, status=status)
        worker = status.Get_source()

        for i, layer in enumerate(range(self.rank, self.num_layers, self.num_masters)):
            tag_base = 2 * layer
            req_w = self.comm.Irecv(nabla_w[i], worker, tag_base + 1)
            req_w.Wait()

            req_b = self.comm.Irecv(nabla_b[i], worker, tag_base)
            req_b.Wait()

        # calculate new weights and biases (of layers in charge)
        for i, dw, db in zip(range(self.rank, self.num_layers, self.num_masters), nabla_w, nabla_b):
            self.weights[i] = self.weights[i] - self.eta * dw
            self.biases[i] = self.biases[i] - self.eta * db

        # send new values (of layers in charge)
        # TODO: add your code

        for layer in range(self.rank, self.num_layers, self.num_masters):
            tag_base = 2 * layer
            self.comm.Isend(self.weights[layer], worker, tag_base + 1)
            self.comm.Isend(self.biases[layer], worker, tag_base)

        self.print_progress(validation_data, epoch)

# gather relevant weight and biases to process 0
# TODO: add your code
# if the master is not 0- send the calculations
if self.rank != 0:
    for layer in range(self.rank, self.num_layers, self.num_masters):
        tag_base = 2 * layer
        self.comm.Isend(self.weights[layer], 0, tag_base + 1)
        self.comm.Isend(self.biases[layer], 0, tag_base)

# if the master is 0- receive **from all other masters** the calculations and update
if self.rank == 0:
    for other_master in range(1, self.num_masters):
        for layer in range(other_master, self.num_layers, self.num_masters):
            tag_base = 2 * layer
            recv_w_req = self.comm.Irecv(self.weights[layer], other_master, tag_base + 1)
            recv_w_req.Wait()
            recv_b_req = self.comm.Irecv(self.biases[layer], other_master, tag_base)
            recv_b_req.Wait()
```

הסבר למימוש חלק 2:

Do worker: נסמן ב- { חלקים שהוספנו לך.

הסבר חלק כחול: לפי ההנחיות, כל העובדים ייחד צריכים לבצע num_of_batches באציגים בכל epoch. לכן, נחלק ביניהם את כמות הבאים.

לאחר חלק זה, נבצע epochים: ניצור בכל אחד את mini_batches עליהם אותו העובד יבצע חישובים, ומחשב גרדיאנים עם forward and back propagation

הסבר חלק צהוב: באירועיות על כל השכבות, נרצה לשלוח כל שכבה לאMASTER שאחראי עליה. מהמשמעות הנתון בdo_master, כל מסטר שלו דרגה master_id, אחראי על כל שכבה i שמקיים mod(num_masters)=master_id(i).

לכן נשלח את הגרדיאנים nabla_b, nabla_w המתאים לכל שכבה. נוסיף תג להודעה כדי ש渴בלת האMASTER מכל ממופות אותה חוזרת לשכבה עבורה התקבלה (ולפרמטרים המותאים), לאחר מכן וכל מסטר אחראי על יתר שכבה אחת. לכן, כל Tag מומפה באופן חד-חד ערכי למספר שכבה, ובעזרת הזוגות שלו ניתן גם לזרה לאיזה פרמטר (w/b) התקבלה ההודעה אצל המאסטר.

הסבר חלק ירוק: בעת עליינו לקבל חוזרת לשכבה מהMASTER האחראי עליה המשקלים והbiases המעודכנים. לכן, ניעזר בכך מייפוי התגים כדי לקבל כל הودעה לbuffer המתאים לה לפ' שכבה, ונכחה לקבללה עט.wait.

Do master

נקצה מקום לגרדיינטים שיתקבלו מכל שכבה עליה אחראי המאסטר, וכן אמצעים לאציגים אן תואם לעובדים.

עת, בכל אפק, בכל batch:

הסבר חלק אדום: קיבל בסטטוס שנשלח לprobe את העובד שרצה לשוחה הודעה לMASTER זהה. אנו מבצעים פעולה זו כדי לאפשר קבלת הודעות מכל עובד שכבר שלח הודעות, מוביל לכפota סדר מסוים בין העובדים מהם קיבל הודעה. לכל שכבה, נחכה לקבלת הודעה לכל buffer מותאים עם wait ובעזרה מייפוי התגים כפי שהסביר.

לאחר חלק זה, יבצעו עדכון למשקלות ו biases עבופר השכבות עליהם אחראי אותו MASTER, בהתאם לערכים שקיבל מהעובד.

הסבר חלק ירוק: נשלח חוזרת לעובד את המשקלות ו biases המעודכנים לשכבות שהוא שלח לחישוב.

הסבר חלק כחול: בסיסם כל העבודה, קיבל מצב בו כל מסטר שומר את התוצאות של החישוב רק עבור השכבות עליו אחראי. נרצה לאגד מידע זה תחת אותו "MASTER אב" שהוא התחילה עם rank 0, וכך נשלח אליו מכל MASTER אחר את כל data שחייב עבור כל שכבה עליה אחראי, ובMASTER עם דירוג 0 נחכה עם wait לקבלת מידע זה.

הערה לחלק 2:

מצין שאנו משתמשים בחלק ב' עבור קריאה וכיתה בקריאות Isend, Irecv MPI שהן קריאות אסינכרניות על מנת לאפשר את התקשרות האסינכרונית הרצiosa.

שאלה 2

הרצה :ring_allreduce sync_network.py משתמשת במימוש

4 cores	8 cores	16 cores
Epoch 1, accuracy 55.9 %.	Epoch 1, accuracy 62.14 %.	Epoch 1, accuracy 50.06 %.
Epoch 2, accuracy 87.36 %.	Epoch 2, accuracy 87.04 %.	Epoch 2, accuracy 85.46 %.
Epoch 3, accuracy 90.16 %.	Epoch 3, accuracy 89.95 %.	Epoch 3, accuracy 90.2 %.
Epoch 4, accuracy 92.05 %.	Epoch 4, accuracy 91.17 %.	Epoch 4, accuracy 91.77 %.
Epoch 5, accuracy 92.79 %.	Epoch 5, accuracy 92.21 %.	Epoch 5, accuracy 92.77 %.
Time reg: 5.122691869735718	Time reg: 5.532649755477905	Time reg: 5.145305156707764
Test Accuracy: 91.95%	Test Accuracy: 91.41%	Test Accuracy: 92.12%
Epoch 1, accuracy 9.83 %.	Epoch 1, accuracy 10.64 %.	Epoch 1, accuracy 10.9 %.
Epoch 2, accuracy 50.85 %.	Epoch 2, accuracy 30.87 %.	Epoch 2, accuracy 20.81 %.
Epoch 3, accuracy 84.93 %.	Epoch 3, accuracy 71.44 %.	Epoch 3, accuracy 46.93 %.
Epoch 4, accuracy 89.19 %.	Epoch 4, accuracy 85.98 %.	Epoch 4, accuracy 80.71 %.
Epoch 5, accuracy 90.09 %.	Epoch 5, accuracy 89.22 %.	Epoch 5, accuracy 88.83 %.
MPICH: Builtin communicator	MPICH: Builtin communicator	MPICH: Builtin communicator
Time sync: 74.23317432403564	Time sync: 79.06123208999634	Time sync: 90.37974667549133
Test Accuracy: 89.1%	Test Accuracy: 88.31%	Test Accuracy: 87.92%
srun -K -c 2 -n 4 --mpi=pmi2 --pty python3 main.py sync	srun -K -c 2 -n 8 --mpi=pmi2 --pty python3 main.py sync	srun -K -c 2 -n 16 --mpi=pmi2 --pty python3 main.py sync

מצורפות הפקודות איתן בוצעה כל הריצה. החלק העליון מציג את הריצה הסינכרונית.

שאלה 3

אנו מראים את הרשת הסינכרונית עם מספר משתנה של עובדים, כאשר לכל ערך הקצנו 2cores.

ניתן לראות כי באימון הרשת הרגילה, הזמן ואחוזי הדיק נשארים יחסית יציבים.

עבור הרשת הסינכרונית, ניתן לראות שכל שמקצים יותר עובדים - הדיק שלה יורד במקצת (גם על פני epochs), והזמן הדרושים לאימון עולה.

הסביר לירידת הדיק: גודל minibatch ברשת הוא 16, ולכן כל עובד בכל אחת מהגרסאות יעבד על minibatch בגודל epochs_of_workers/16. לעומת האימון משתמש ב4 עובדים יותר_epochs_of_workers/4, והוא הכיל גדול מבן אחר minibatches בשאר הגרסאות.

מכיוון שכל עובד מחשב גרדיאנטים על חלק המידע (minibatches) שהזקקה לו, וזה מחושב ממוצע על פני כל העובדים, נקבל שכל שагדים אנטיים מחושבים על קבוצה קטנה יותר של דוגמאות- קר' יש פחתה הכללה בחישוב (ובחישוב הממוצע שבתבצע לאחר מכן), וכןן ההתקנסות איטית יותר, ונקבל עבור אותו מספר epochs דיק נמוך יותר. ניתן לראות זאת בבירור עבור דיק שמופרש לכל epoch.

הסביר לעלייה בזמן האימון: הזמן שנדרש לאימון עולה מכיוון שככל שיש יותר עובדים, יש בינם יותר תקשורת (מערכת עמוסה כי שקצבות פיק בהן צריכה להתבצע הרבה תקשורת) וכן אנו צריכים לבצע יותר סינכרונים (ובפרט- יותר הפיצות מידע בין כל העובדים עבור הepochs), דבר שולח לנו בזמן החישוב (כלומר: למורת המקובל בחישוב, אנחנו מקבלים תקורה ממשוערת בתקשורת שצריכה להתבצע באופן סדרתי).

השוואה עם האימון המקורי: באימון המקורי, הגרדיאנטים מחושבים על ידי תהליך אחד, וכך אין תקשורת ואין סינכרון, דבר שלא מאט את עבודות האימון. כמו כן, הגרדיאנטים מחושבים על קבוצת הדוגמאות (minibatch) hic גודלה מבן כל ההרצות, הכוללת 16 דוגמאות, וכך ההתקנסות של האימון ברשת זו מהיר יותר מהגרסאות האחרות ונקלע עבורו דיק גבוה תחת אותו מספר epochs.

שאלה 4

האלגוריתם הסינכרוני מקבל speedup לשני העקרונות של האmdl: זאת מכיוון שהוא גודל הבעה קבוע, והחלק הנINITן למקובל קבוע. וכך, החלקים הסדרתיים בתוכנית (התקשורת והסינכרונים) מגבלים את ההאצה ואף מאטים את התוכנית ככל שכמות העובדים גדלה.

כדי לקבל speedup לשני גוסטבסון, علينا לבצע scaling: נרצה להגדיל את הרשת עם הגדלת כמות העובדים, ואת כמות הדוגמאות בחילוק minibatch עליהן מבוצע האימון, על מנת שהחלק הסדרתי לחוטין בתוכנית (שוב, הסינכרון והתקשורת) ישאף ל-0 ככל שהבעה גדלה.

שאלה 5

הרצה :async_network.py

הרצה מקורית	2 masters, 4 cores	2 masters, 8 cores	4 masters, 8 cores	4 masters, 16 cores
יחסית זהה לכל כמות עובדים ומאסטרים	2 workers, 2 masters	6 workers, 2 masters	4 workers, 4 masters	12 workers, 4 masters
לפי כל פקודות ההרצה	srun -K -c 2 -n 4 --mpi=pmi2 --pty python3 main.py async 2	srun -K -c 2 -n 8 --mpi=pmi2 --pty python3 main.py async 2	srun -K -c 2 -n 8 --mpi=pmi2 --pty python3 main.py async 4	srun -K -c 2 -n 16 --mpi=pmi2 --pty python3 main.py async 4
Epoch 1, accuracy 56.3 %. Epoch 2, accuracy 86.35 %. Epoch 3, accuracy 89.44 %. Epoch 4, accuracy 91.16 %. Epoch 5, accuracy 92.18 %. Time reg: 5.4892418384552 Test Accuracy: 92.18%	Epoch 1, accuracy 9.61 %. Epoch 2, accuracy 9.61 %. Epoch 3, accuracy 9.61 %. Epoch 4, accuracy 9.61 %. Epoch 5, accuracy 9.61 %. MPICH: Builtin communicator 44 Time async: 3.4467759132385254 Test Accuracy: 92.03%	Epoch 1, accuracy 10.09 %. Epoch 2, accuracy 10.09 %. Epoch 3, accuracy 10.09 %. Epoch 4, accuracy 10.09 %. Epoch 5, accuracy 10.09 %. MPICH: Builtin communicator 44 Time async: 2.567652940750122 Test Accuracy: 11.35%	Epoch 1, accuracy 9.15 %. Epoch 2, accuracy 9.15 %. Epoch 3, accuracy 9.15 %. Epoch 4, accuracy 9.15 %. Epoch 5, accuracy 9.15 %. MPICH: Builtin communicator 44 Time async: 2.474914312362671 Test Accuracy: 88.24%	Epoch 1, accuracy 9.67 %. Epoch 2, accuracy 9.67 %. Epoch 3, accuracy 9.67 %. Epoch 4, accuracy 9.67 %. Epoch 5, accuracy 9.67 %. MPICH: Builtin communicator 44 Time async: 2.474914312362671 Test Accuracy: 9.8%

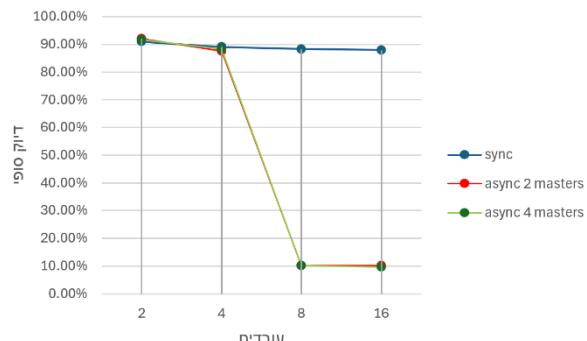
שאלה 6

תחילה, ניתן לראות שהרצה על ליבות רבות יותר מובילה להאצה בчисלוב. זאת מכיוון שעבודת כל העובדים לא תליה אחת בשניה, ככלומר כל עובד מבצע את עדכונו הגרדיינט שלו, מקבל מהMASTER גרדיאנטים חדשים, ומשדר בעבודה שלו. יש משמעויות פחות סync'רונים מאשר בשיטה הקודמת (במקום לסכם בין כל העובדים, לשנכרן בין עובד ומהMASTERים שלו), ויש מתקבל הרבה- גם בчисלובי הגרדיאנטים (batch) בין העובדים ומחושב במקביל מבלי להסתמך (כל MASTER מעדכן פרמטרים לשכבות עליין הוא אחראי, וזה קורה במקביל עבורו).

עם זאת, ניתן לראות ירידheid בבדיקה האימון עם אותו מספר epochs ככל שיש יותר עובדים. זאת מכיוון שהפרמטרים שאיתם העובד המשמש לחישוב הגרדיינט ואז מחזיר את התוצאה לעובדים- לא בהכרח עדכנים- בזמן שעבוד זה מחשב את הגרדיאנטים, העובדים אחרים מס'ימים את העבודה שלהם וועשים עדכנים לקבוצת הפרמטרים. כפי שראינו בהרצאה, GAP (push远程参数更新) גודל כל שיש יותר עובדים, מה שmobiel לכך שרלוונטיות הגרדיאנטים שנדוחפים בשלב ההpush פחות טובה, והדיק בAIMON יורד. בעיה זו נקראת gradient stainless.

שאלה 7

אם נשימוש בparameter server יחיד, תהיה בעיית סקלibility: ככל שנרצה מערכת גדולה ומהירה יותר, כך יווצר עומס על שרת הפרמטרים כי כל העובדים יפנו אליו לעדכנים, והוא יփוך להיות צואר בקבוק ויאט את המערכת. בעיה זו נקראת sharding. לכן, נרצה לחלק את העבודה בין כמה SERVER פרמטרים כך שכל חלק של הפרמטרים ייחשך על ידי שרת שונה, מה שיפחית את העבודה מכל שרת.

שאלה 8**שאלה 9**

המיומש האסינכריוני יורד בדיקו בכל שכמות העובדים עולה בגלל בעיית hessian stainless. עליה פירטונו בשאלה 6 ☺

שאלה 10

נשווה בין הגישה הסינכרונית לגישה האסינכרונית.
היתרון בגישה הסינכרונית הוא שהוא מספקת דיק גודל יותר, מכיוון שתמיד כל העובדים עובדים על קבוצת פרמטרים עדכניים.
החיסרון בשיטה זו היא שכל שיש יותר עובדים, אנחנו צריכים לבצע יותר תקשורת בנקודות ספציפיות, מה שגדיל את החלק הסדרתי לביצוע, ומאט את המערכת (כפי שראינו, האטה אפיו ביחס לביצוע סדרתי רגיל)
כמו כן, ראיינו בהרצאה את הבעיה הבאות:

בעיה בסקלביות של השיטה הסדרטיבית: אם נגדיל את הרשות, כל עובד יצטרך לעבוד על batch גדול יותר, מה שייגע בדיק הסופי. בנוסף, הגדלת הרשות גורמת לכך שיש לבצע יותר תקשורת בנקודות הסינכרון, מה שוביל למערכת עומס בתחילה וסוף כל איטרציה.
בעיה בחישוב מעלה ענן: יתכנו משאבי ענן שישולים להיות איטיים יותר כי הם משותפים עם אפליקציה שצריכה הרבה מהם, ולכן חלק אימון שישתמש במשאב זה כעובד ייחודה תואמת ביחס לחלקים האחרים של האימון, מה שיאט עוד יותר את האימון הכללי.

היתרון בשיטה האסינכרונית הוא שהמקבול גבוי, ולכן אנחנו נקבל האצה בזמן הנדרש לאימון כל שיש מספר גודל יותר של עובדים (ומאסטרים המתאים להם מבוקן, אחרית ייה sharding). עם זאת, החיסרון הוא שבגלל בעיית השדרה, הדיק ירד בצורה משמעותית (כלומר - זו בעית סקלביות).

שאלה 11

2 cores	4 cores	8 cores
Testing array size: 4096 Naive all-reduce time: 0.0027933120727539062 Ring all-reduce time: 0.001367330551147461 Comparing results... Naive all-reduce correct: True Ring all-reduce correct: True Testing array size: 32768 Naive all-reduce time: 0.011008024215698242 Ring all-reduce time: 0.006892681121826172 Comparing results... Naive all-reduce correct: True Ring all-reduce correct: True Testing array size: 262144 Naive all-reduce time: 0.07940196990966797 Ring all-reduce time: 0.03997087478637695 Comparing results... Naive all-reduce correct: True Ring all-reduce correct: True Testing array size: 2097152 Naive all-reduce time: 0.6528196334838867 Ring all-reduce time: 0.3086974620819092 Comparing results... Naive all-reduce correct: True Ring all-reduce correct: True	Testing array size: 4096 Naive all-reduce time: 0.0071108341217041016 Ring all-reduce time: 0.008584976196289062 Comparing results... Naive all-reduce correct: True Ring all-reduce correct: True Testing array size: 32768 Naive all-reduce time: 0.02593851089477539 Ring all-reduce time: 0.03288674354553223 Comparing results... Naive all-reduce correct: True Ring all-reduce correct: True Testing array size: 262144 Naive all-reduce time: 0.4475996494293213 Ring all-reduce time: 0.3313441276550293 Comparing results... Naive all-reduce correct: True Ring all-reduce correct: True Testing array size: 2097152 Naive all-reduce time: 3.6008450984954834 Ring all-reduce time: 0.6259067058563232 Comparing results... Naive all-reduce correct: True Ring all-reduce correct: True	Testing array size: 4096 Naive all-reduce time: 0.015838146209716797 Ring all-reduce time: 0.00944972038269843 Comparing results... Naive all-reduce correct: True Ring all-reduce correct: True Testing array size: 32768 Naive all-reduce time: 0.06008028984069824 Ring all-reduce time: 0.013703344252441406 Comparing results... Naive all-reduce correct: True Ring all-reduce correct: True Testing array size: 262144 Naive all-reduce time: 0.5364594459533691 Ring all-reduce time: 1.1350955963134766 Comparing results... Naive all-reduce correct: True Ring all-reduce correct: True Testing array size: 2097152 Naive all-reduce time: 4.322660446166992 Ring all-reduce time: 8.817091941833496 Comparing results... Naive all-reduce correct: True Ring all-reduce correct: True

ניתן לראות שמיון all reduce ring all reduce מהיר יותר מאשר naive. זאת מכיוון שבמימוש הנאייבי אנחנו שולחים את כל המערכת לכל התהיליכים, ככלומר יש הרבה תקשורת, ובמימוש של ring יש לנו שולחים חלקים מהמערכת לחץ מהתהיליכים, ולכן סך התקשורת נמוך יותר, דבר שמאפשר למערכת לעבוד מהר יותר ויעיל יותר.

נתיחס ל-2 הבעיות האחרונות ב-8 cores: כפי שניתן לראות בהן, דיק האמימוש הנאייבי מהיר יותר פי 2. זאת מכיוון שעם יותר עובדים, יש יותר סבבי תקשורת שצרכיים להתבצע, ומכוון שהמערכות מארוד גדולים בRICTות אלו, זה יוצר עומס בתקשורת.

שאלה 12

ניתוח סיבוכיות למימוש הנאייבי: בהינתן N תהיליכים DATA1 לשילוח בגודל M מכל אחד, כל תהיליך שולח $N-1$ הודעת בגודל M . לכן סיבוכיות המידע הנשלח מטה תהיליך אחד היא $(MN)^0$, וכך הסיבוכיות של כל המידע הנשלח מכל התהיליכים היא $N \cdot O(MN^2) = O(MN^2)$.

שאלה 13

ניתוח סיבוכיות ל`reduce_all`: בהינתן N תהליכי DATAI לשילוח בגודל M מכל אחד, כל תהליכי שולח פעמיים (פעם אחת לחישובים ופעם נוספת להפצה) ב- $1-N$ איטרציות $\frac{M}{N}$ פיסות מידע, וכן סיבוכיות המידע הנשלח לתהליכי הוא $(M) = (N \times \frac{M}{N})^0 = 0$. וכך הסיבוכיות של המידע הנשלח מכל התהליכים הוא $(MN) = 0 \times N = 0$.

שאלה 14

א. Seq. consistency => coherent causal consistency

הוכחה: הוכחנו בתרגול בשקף 26 ש $\text{seq. consistency} \Rightarrow \text{causal consistency}$ -
כמו כן, באותו השקף ראיינו $\text{seq. consistency} \Rightarrow \text{coherence}$

לכן תוכנית P שחוקית תחת seq. causal ותחת coherence , וכך הכל לפי הגדרת המודל בשאלת היא חוקית גם תחתו.

ב. נפריר על ידי דוגמה נגדית:

Process 1	Process 2
Read y,1	Read x,2
Write x,2	Write y,1

התוכנית חוקית תחת coherence

נראה שלכל משתנה מסוים, קיים סידור חוקי לקרואות ולכתיבות.

סדר חוקי לא: $y,1 \rightarrow x,2 \rightarrow x,2 \rightarrow y,1$

סדר חוקי לע: $x,2 \rightarrow y,1 \rightarrow x,2 \rightarrow y,1$

התוכנית חוקית תחת causal

אין כתובות לאוטו המשטנה בין שני התהליכים, ולכן הכתובות הן לא בלתי תלויות, וכל סידור שלתן חוקי כי אין בכראח שככל התהליכים יראו אותן באותה סדרה.

לכן התוכנית חוקית גם כן תחת הגדרת המודל החדש בשאלת, תחת coherent causal consistency.

עם זאת, התוכנית לא חוקית תחת seq. consistency除非 קיימים סידורים לקרואות ולכתיבות שקיים סטטוטני עם שני התהליכים (ראיינו עבור דוגמה זו בתרגול).