



אוניברסיטת בן גוריון בנגב

הפקולטה למדעי ההנדסה

בית הספר להנדסת חשמל ומחשבים

דו"ח לפרויקט מסכם בקורס "תכנון רשתות מחשבים" – 371.1.0281

Performance Optimization of Multi-Stage Queue System Using OMNeT++

מגישים : מאיה שמחי 207487653

דולב איזנברג 208212845

מספר קבוצה : 4

תאריך הגשה : 6/6/2025

תוכן עניינים

31. מבוא
52. מידול המערכת
93. יודי הסימולציה
11.....	.4. מימוש הסימולציה
18.....	.5. בדיקת שפויות
20.....	.6. תוצאות – סימולציה סטטistica
20	6.1. סימולציה סטטistica : מצב עבודה רגיל – 3 Scanners
29	6.2. סימולציה סטטistica : מצב עבודה עמוס – 5 Scanners
37.....	.7. תוצאות - סימולציה דינמית
38	7.1. סקר ביצועים – עליה לינארית, ירידה עפ"י היסטוריזיס
53	7.2. סקר ביצועים – שיעורן זמן לריקון התור (Queue Drain-time Estimation)
62	7.3. סקר ביצועים – השוואת בין המנגנונים השונים
64.....	.8. סיכום ומסקנות

1. מבוא

הפרויקט הוא **במראה אמיתית** שיש בمكان העבודה של מאיה, שם סורקים ומנתחים את המידע של הלקחות כדי לחת להם מידע איפה מוחזק אצלם מידע פרטי או בעיתוי מבחינת אבטחת מידע, רגולציה שונות וכו'.

בין התהליכים השונים שמתבצעים בחברה, קיימת מערכת תלת-שלבית שאחראית לעבד בקשנות מלכותות. המערכת מבוססת/משתמשת ב-Q-MQ – תוכנת Open-source לניהול הודעות (כתובה ב-Erlang), המשמשת בתור "חוצץ"/"מגשר" (intermediary) לקבלת הודעות מהיצרנים ומעבירת אותם אל היצרנים. זה נעשה על ידי שימוש בתורים (queues), בהם מאוחסנות הודעות עד שהן יוכלו להיצרך ע"י היצרנים. RabbitMQ עשויה שימוש ב프וטוקול AMQP (Advanced Message Queueing Protocol), פרוטוקול סטנדרטי לתורים-בסיסי-הודעות.

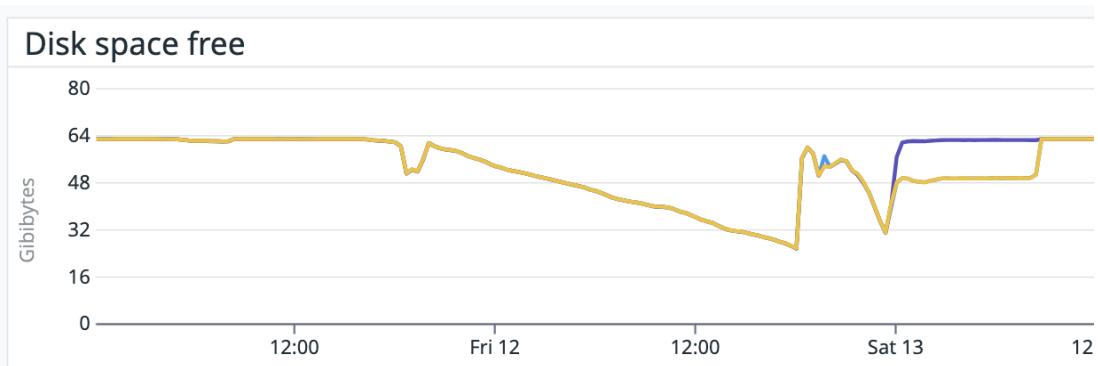
אופן העבודה של המערכת:

1. רכיבים (תוכנתיים) הנקראים Scanners סורקים את המידע של הלקחות ומיצרים בקשנות עיבוד גולמית שונות בהתאם לצרכי הלקחות.
2. בהתאם לסוג הבקשה וסט חוקים (שלא נרחב עליו), ה-Scanners מעבירים את הבקשות לתורים ייעודיים שמנוהלים ע"י RabbitMQ. אנחנו נתמקד בסוג ספציפי של בקשנות שכולן מועברות לתור יחיד.
3. הבקשות הגולמיות נשלפות מן התור אל ה-Orchestrators – רכיבים שביצעים עיבוד מקדים ומהיר באופן ייחסי, ומוכנסות כולם לתור נוסף, שמנוהל גם הוא ע"י RabbitMQ. כל Orchestrator יודע לטפל בבקשת אחת בכל רגע נתון.
4. הבקשות המעובדות מטופלות ע"י Catalogs, שמשיכים את הטיפול בבקשת. משך טיפול זה הינו ארוך יחסית.

במערכת שנמצאת כרגע בשימוש ב-BigID, כמוות היחידות בכל שלב משתנה באופן דינامي בהתאם לעומס העבודה, ואחת הביעות היא שכאשר המערכת תחת עומס, התורים המנוהלים ע"י RabbitMQ מגיעים למוגבלה שלהם וקורסים. כשקרישה כזו מתרחשת, נוצרת תגובת שרשרת שגורמת לקריישה של כל המערכת, וכל התוכן (גם זה שאינו קשור לתורים שהובילו לקרישה) נאבד. ורידיה זו בתפוקה (ואיבוד של המידע שנמצא בעבודה) גורם לבזבוז משאביים, זמן וכיסף.



אייר 1.1 : צילום גף-h Consumers במערכת האמיתית. ניתן לראות את נפילות המערכת כאשר אין Consumers.



אייר 1.2 : גראף מקום האחסון במערכת האמיתית. ניתן לראות קriseה מלאה של המערכת והיעלמות של כל ההודעות (פינוי פתאומי של זיכרון שהכיל הודעות בעת קriseה המערכת).

Datadog APP 12:29 AM

Triggered: Found bigid-data-catalog crash on cluster_name:production-mt-3-us-east-1,kube_deployment:bigid-data-catalog-consumer,kube_namespace:unum @slack-bigid-data-catalog-crashes

```
avg(last_30m):default_zero(sum:kubernetes.containers.state.waiting{cluster_name:p*, reason:crashloopbackoff, !kube_namespace:staging*, kube_deployment:bigid-data-catalog*} by {kube_namespace,cluster_name,kube_deployment}) > 0.5
```

Metric value: 0.544 (7 kB) ▾

Tags	Notified
cluster_name:production-mt-3-us-east-1, @slack-bigid-data-catalog-crashes	
kube_deployment:bigid-data-catalog-	
consumer, kube_namespace:unum	

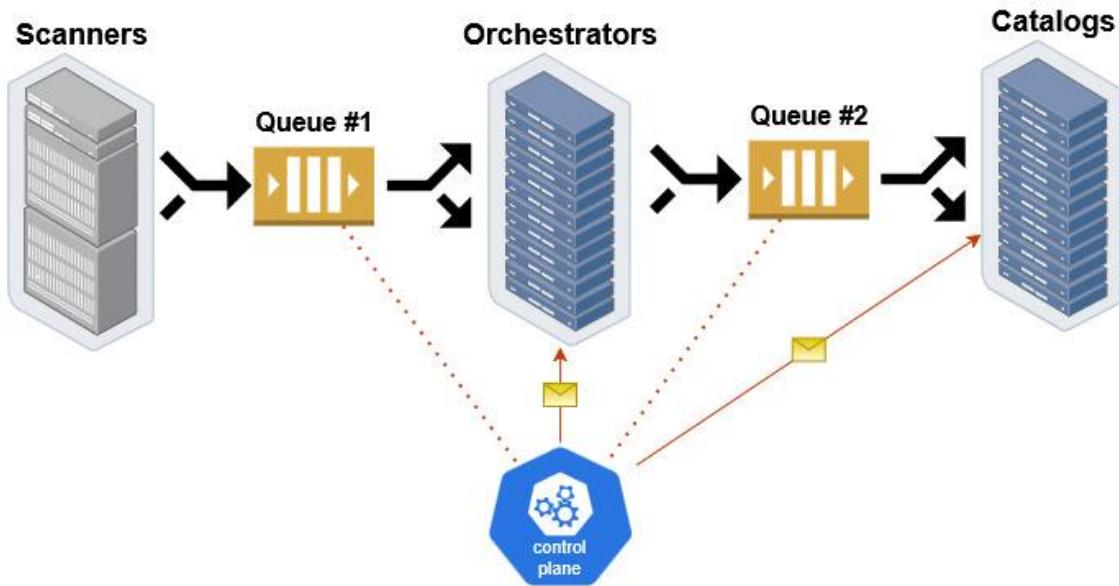
Mute Monitor **Declare Incident**

אייר 1.3 : הודעה שגיאה ב-Slack על קriseה המערכת.

בכדי לטפל בבעיה זו, נרצה למדל את המערכת באמצעות OMNet++, ובאמצעותה למצוא את כמויות היחידות המותאמות והיחסים ביניהן (בהתאם לעומס על המערכת) בכדי למנוע את קriseה התורמים ולדואג שהמערכת תעבור במצב חלקה ותתאים את עצמה לעומס בו היא נתונה.

2. מיזוג המערכת

במבט על, המערכת מורכבת מ-3 חלקים עיבוד מרכזיים – Catalogs, Orchestrators ו- Scanners (כאשר מכל אחד קיימים מספר יחידות, שהניהול שלהם מתבצע ע"י יחידת בקרה בהתאם לעומס בתורים שבמערכת. בין חלקים אלו חוצים 2 תוררים (המנוהלים ע"י RabbitMQ) שמקבלים מידע מהשכבה הקודמת ומפזרים אותו בין הזריםים בשכבה הבאה.



אייר 2.1 : שרטוט ממבט-על של מודל המערכת.

נציג את אופן העבודה הכללי של כל אחד מהרכיבים פה, ואת האופן בו בחרנו למודל אותם.

Catalogs .2.1

במערכת המשמשת: כל קטלוג מקבל הודעה/עבודה (Job) אחת בכל פעם, מבצע אותה, וכאשר הוא מתפנה הוא מבקש מהתור הודעה/עבודה נוספת. ניהול חלוקת העבודה בין Catalogs השונים נעשה ע"י ה-RabbitMQ שאחראי גם לניהול התור.

קצב ביצוע העבודות אינו אחיד, ומຕוך נתוני המערכת האמיתיים ניתן לחשב קצב צריכה של $0.2 \frac{Jobs}{Sec}$. בממוצע, ככלומר 5 שניות לבצע עבודה אחת.



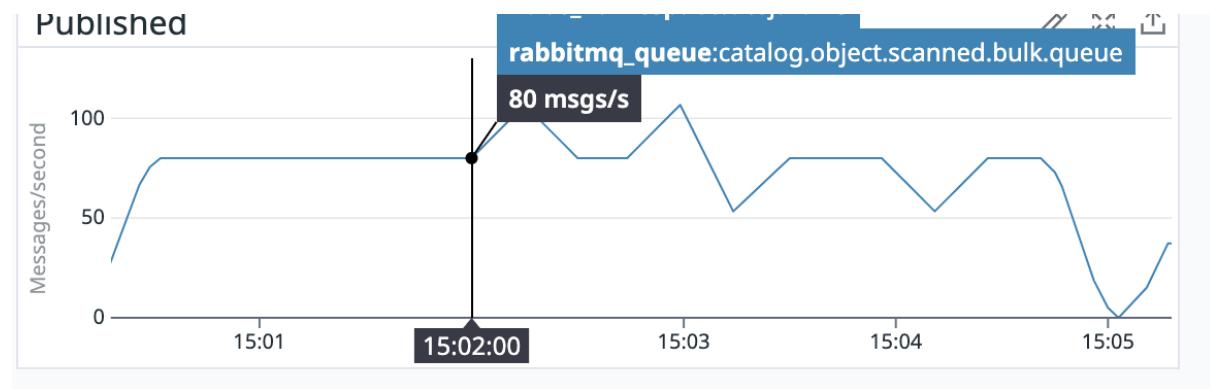
אייר 2.2 : גרפ ביצוע עבודות ע"י ה-Catalogs במערכת.

במודל שלנו : כדי שקצב הביצוע לא יהיה קבוע, נmdl אותו בתור מ"א אקספוננציאלי עם תוחלת של 5 שניות (Exponential(5s)). כאשר ה-Catalog אינו מבצע עבודה, הוא שולף את ההודעה שנמצאת בראש התור (של העבודות המחכות לביצוע). אם ה-Catalog פנו והתור ריק, הוא מבצע polling לתור באופן אקראי עם תוחלת של 0.01 שניות.

Orchestrators .2.2

במערכת המעשית : ה-Orchestrators פועלים באופן דומה ל-Catalogs, כאשר הם מקבלים עבודות מתור שמיינים ה-Scanners, מבצעים עיבוד על המידע ומכוונים את התוכרים שלהם לתור נוסף.

ל-Orchestrators זמן עיבוד קצר יותר משמעותית מאשר ה-Catalogs, והוא ממוצע עומד על $\frac{Jobs}{Sec} = 4$. כלומר, 0.25 שניות עיבוד לכל עבודה.



אייר 2.3 : גרפ קצב ביצוע עבודות ע"י ה-Orchestrators. בזמן הצילום היו כ-20 יחידות פעילות, על כן-

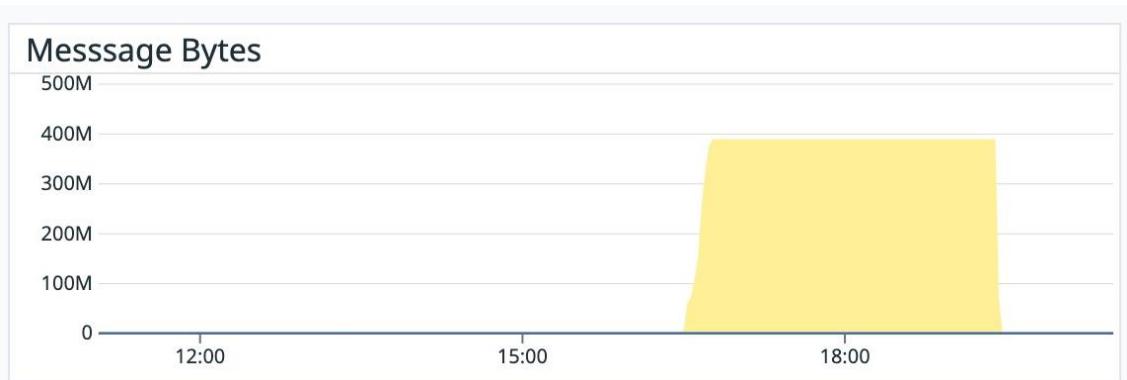
$$\frac{80}{20} = 4 \text{ } \frac{\text{Msgs}}{\text{Sec}}$$

במודל שלנו : נmdl את קצב ביצוע העבודות בתור מ"א אקספוננציאלי עם תוחלת של 0.25 שניות (Exponential(0.25s)). כאשר יחידה מסוימת מבצע עבודה, היא מכניסה את העבודה לתור (שמננו ה-

শولפים), ושולפת את העבודה שנמצאת בראש התור (שםוזן מה-*Scanners*). אופן שליפת העבודות מהתור זהה לשולפ *Catalogs*.

Queues.2.3

במערכת האמיתית: התורים, שמנוהלים ע"י *RabbitMQ* וחוצים בין ה-*Catalogs* ל-*Orchestrators* ובין ה-*Scanners* ל-*Orchestrators* מקבלים עבודות מכלל היצרנים (*Orchestrators/Scanners*), תלוי באיזה תור מדובר), מתחסנים אותו ומחלקים את העבודות האלה בהתאם לחוקים פנימיים ולזמינים היצרנים, שמודיעים כאשר הם זמינים לבצע עוד עבודה. המערכת יש נפח אחסון מוגבל בתור שמושפע מתוכן העבודות.



אייר 4 : נפח האחסון של העבודות בתור במערכת.

במודל שלנו : במקומות שחלוקת העבודה תעשה ע"י התורים עצמם והיצרנים מודיעים על זמינים (והתור מגיב לך ע"י שליחת עבודה לאוטו צרכו) - היצרנים פונים שירות אל התור ושולפים מראש התור עבודה.

כדי למדל את זמן העבודה הפנימי וההשניה שנוצרת בתוך *RabbitMQ*, אנחנו מוסיפים לתור זמן עיבודAKERAI בערך תוחלת של *1ms* לכל הודעה שמגיעה אליו (נועד למדל את הזמן שלוקח *RabbitMQ* לקבל את המידע ולבצע את החישובים הפנימיים שלו).

כדי למדל את מגבלת נפח האחסון, הערכנו באמצעות נפח האחסון בו התור קורס את כמות ההודעות שנמצאת בו ברגע הקriseה, וערך זה נע בין 100-120 הודעות. על כן, אנחנו נחמיר ונגדיר מגבלת אורך לתור שלנו של 100 הודעות.

Scanners.2.4

במערכת האמיתית: מנגנון ה-*Scanners* הוא חלק מלוגיקה נרחבת הרבה יותר (יחידות אלו מבצעות פעולות נוספות). לצורך הבעה שלנו, ניתן להניח ה-*Scanners* מייצרים עבודות בקצב של 0.1 שניות לעבודה.

במודל שלנו : נמדל את קצב ייצור העבודות בתור מ"א אקספוננציאלי עם תוחלת של 0.1 שניות. את העבודות האלה ה-*Scanners* יזינו לתור שבמהמשך מעביר ל-*Orchestrators*.

במערכת הקיימת, כמוות היחידות המקסימליות הן :

כאשר המערכת מגיעה לעומס כזה באופן שאיינו רגעי, התורים קורסים ואיתם כל המערכת.

נמחיש זאת ע"י הרצה קצרה של הסימולציה שלנו (עליה נפרט מיד):

Experiment	Measurement	Replication Module	Name	Value
General	#0	TestNetwork.queue	queueLength:timea	~10.9407
General	#0	TestNetwork.queue	queueLength:max	36
General	#0	TestNetwork.queue	queueLength:avg	~10.6143
General	#0	TestNetwork.queue	dropped:count	0
General	#0	TestNetwork.queue2	queueLength:timea	~109.106
General	#0	TestNetwork.queue2	queueLength:max	284
General	#0	TestNetwork.queue2	queueLength:avg	~110.475
General	#0	TestNetwork.queue2	dropped:count	0

טבלה 2.2: ערכי התורים בסימולציה עבור הרצה של 5 שניות, ללא הגבלת אורך התור.

כפי שנitinן לראות, תור מס' 2 (queue2), שבין ה-Orchestrators ל-Catalogs, מתנפח לאורך מקסימלי של 284 ואורך ממוצע של 110.475, ועל כן גם בסביבת הסימולציה אנחנו יכולים לשחזר את נפילת המערכת.

Controller 2.5

במערכת האמיתית: מנגנון הבדיקה מוסת את כמות הרכיבים שעובדים בכל רגע נתון במערכת. הוויסות מתבצע ע"י דגימה אחת לשניה של אורך התורים, באופן הבא:

$$\text{Number of elements} = (\text{Queue length}) * \frac{\text{Maximal # of pods} - \text{minimal # of pods}}{100}$$

אם תוצאה החישוב הנ"ל גדולה מכמות הרכיבים שמופעלים – מעלים את כמות הרכיבים לערך שהתקבל.

אם הכמות קטנה – ממצאים את הדגימות ארבע השעה האחידנה ומורידים לערך התואם עפ"י הנוסחה.

במודל שלנו: נmdl את אופן העבודה הנ"ל:

- בכל X שניות תבוצע דגימה של אורך התורים ע"י גישה למודול של התורים ע"י הAKER.
- לאחר דגימה, מנגנון הבדיקה יעשה חישוב ויחליט האם להדליך או לבנות יחידות נוספות (בקטלוגים והמתאימים באופן נפרד).
- הדלקה וכיבוי רכיבים ייעשה על ידי שליחת הודעת הדלקה/כיבוי לרכיבים הספציפיים.

3. יעדוי הסימולציה

בקצהה: נמצא את כמות היחידות השונות (Scanners, Orchestrators & Catalogs) בעומסים שונים של המערכת כך שהתורים שלנו לא יתמלאו ונקבל את הביצועים הטובים ביותר (סימולציה סטטית). לאחר מכן, נשלב מנגנון בקרה המאפשר לשולט על כמות הרכיבים הפעילים במערכת ונבדוק 2 פונקציות בקרה שונות, עליהם נבצע סקר ביצועים על הפרמטרים שבסליתנותנו כדי למצוא את הקונפיגורציות האופטימליות. לבסוף, נשווה את הקונפיגורציה הנ"ל עם גישות אחרות לביקורת המערכת (fonkציות שונות) שלדעינו יכולות לתת ביצועים טובים יותר.

3.1. שלב א' – סימולציה סטטית:

נתמקד ב-2 תרחישים :

1. מצב עבודה רגיל – פועלים 3 Scanners
2. מצב עבודה עמוס – פועלים 5 Scanners

את כל התרחישים נריץ בתנאים הבאים :

- אורך הסימולציה הוא כ-10,000 שניות (~3 שעות) – מאפשר לבצע stress testing ולקבל תוצאות אמינות יותר.
- הקיבולת המקסימלית של כל אחד מהתורים הינו 100 הודעות (+הודעה אחת שנמצאת 'בדרך').
- קצב העבודה של כל היחידות במערכת הון כפי שמידלנו קודם קודם עפ"י הקצבים הקיימים במערכת האמיתית.

היעדים שלנו :

1. אפס קריסות של התורים – כמובן, התור אינו מגע למגבלה (מתבטא באיבודי עבודות, dropped bytes, כאשר התור מלא). נשאף להגעה מקסימלית למחצית מגודל התור (עד 50).
2. השהייה (Latency) נמוכה ככל הניתן (השהייה ממוצעת ומקסימלית).
3. כמות הרכיבים (Orchestrators, Catalogs) המינימלית שנדרשת ביצועים מספקים.

3.2. שלב ב' – סימולציה דינמית

نبזק את האפשרות להשתמש במערכת בקרה דינמית כדי לשולט בכמות המתאימים והקטלוגים הפעילים, במטרה להשיג ביצועים טובים במש' מקרים תוך כדי הפחיתה בכמות היחידות המופעלות.

נשתמש בתוצאות הסימולציה הסטטית בתור ערכי בסיס למספר המקסימלי של היחידות אליהן המערכת יכולה להגיע ונתקנן 2 פונקציות בקרה :

1. עליה לינארית, ירידה עפ"י היסטורי (Drain estimation)
2. שערוך זמן לריקון התור (Drain estimation)

פירוט מكيف על אופן פעולה הפונקציות נמצא בפרק 4.5 (אופן עבודה כללי) ו- 7 (לכל פונק' ספציפית)

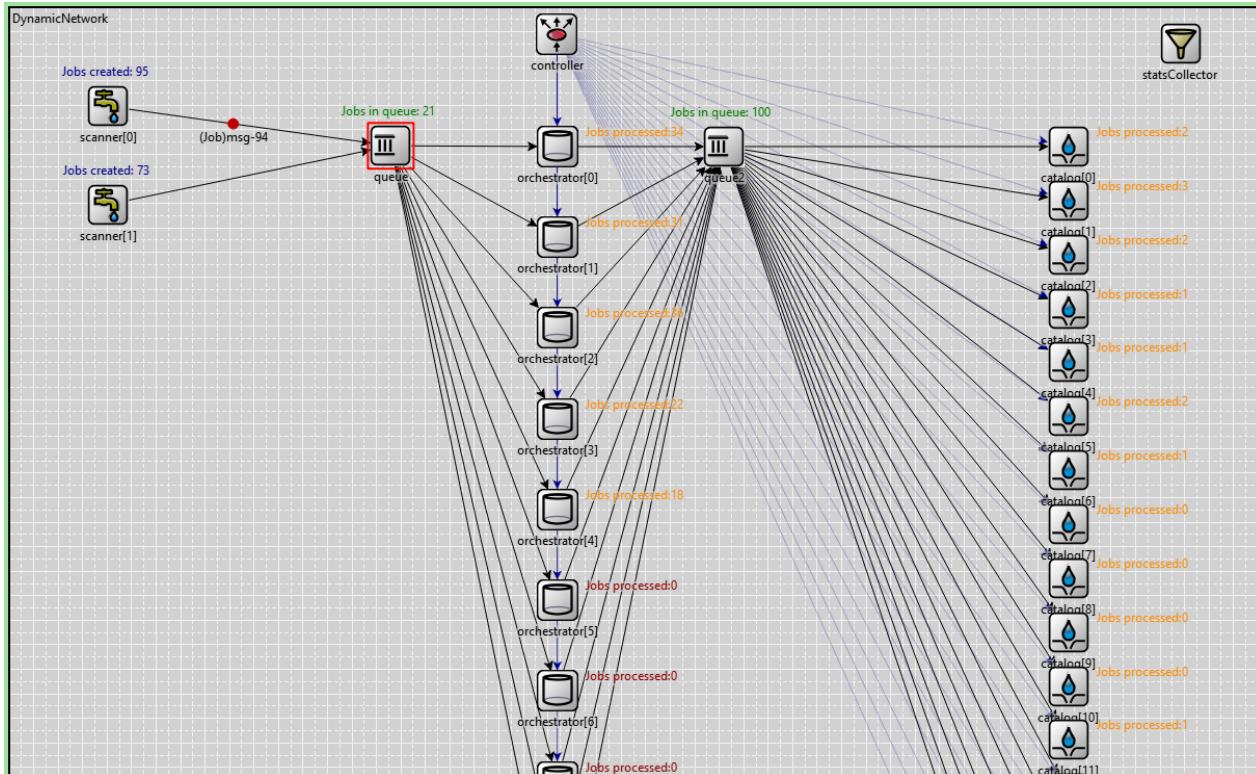
מבצע סקר פרמטרים על פרמטרים שבשליטתנו בשתי הפונקציות האלו ונמצא את קומבינציית הפרמטרים האופטימלית לפי מס' מדדים שימושיים אותן :

- כמות הרכיבים הפעילים
- השהיה (Latency)
- תפוקה (Throughput)
- תנודתיות המנגנון (כמו 'פתיחה' ו'סגירות' של רכיבים מתבצעות לאורך הסימולציה)
- איבודי הודעות (כ-% מכלל ההודעות)

ולבסוף נשווה בין הקומבינציות האופטימליות של שתי הפונקציות האלו ותוצאות הסימולציה הסטטיסטית.

4. מימוש הסימולציה

הסימולציה מכילה 28 פרמטרים שונים לשיליטה על התנהלות המערכת, ואיסוף סטטיסטיות על 13 נתונים שונים (בצורה סקלרית, וקטורית וכו').



אייר 4.1 : GUI הסימולציה הדינמית בזמן עבודה.

הסימולציה מורכבת מהמודולים (Classes) הבאים:

1. מודול תור – Queue.cc
2. מודול סורק – Scanner.cc
3. מודול מתאם – Orchestrator.cc
4. מודול קטלוג – Catalog.cc
5. מודול ייח' בקרה – Controller.cc – (בסימולציות הדינמיות)
6. מודול איסוף סטטי מרוכז – StatsCollector.cc

בנוסף, יצרנו קובץ הودעה ייעודי (Job.msg) המאפשר להודעתה במערכת להכיל מידע שנדcatch לדעת על ההודעות – זמן השהייה בתור, זמן התחלתה (מתי ההודעה נוצרה) וזמן הגעה לתור :

```
message Job {
    simtime_t totalQueueingTime;
    simtime_t startTime;
    simtime_t queue_arrivalTime;
}
```

אייר 4.2 : הגדרת קובץ ההודעה לסימולציה.

לכל אחד מمبرכי הסימולציה – הսטטיטית והדינמית, יש קובץ ned. שמנדר את חיבוריו הרשות המתאימים אליו, כאשר כמוות היחידות מכל רכיב (קטלוגים, מתאימים וسورקים) היא פרמטר שמנדר בקובץ ה-.ini.

```

connections:
  for i=0..numScanners-1 {
    scanner[i].out++ --> queue.in++;
  }
  for i=0..numOrchs-1 {
    queue.out++ --> orchestrator[i].in++;
    orchestrator[i].out++ --> queue2.in++;
  }
  for i=0..numcatalogs-1 {
    queue2.out++ --> catalog[i].in++;
  }
  for i=0..numOrchs-1 {
    controller.orch_out++ --> { @display("ls=darkblue,0.25"); } --> orchestrator[i].in++;
  }
  for i=0..numcatalogs-1 {
    controller.catalog_out++ --> { @display("ls=darkblue,0.25"); } --> catalog[i].in++;
  }
}

```

.Ayar 4.3 : החיבורים בקובץ הקינפוג של הרשות הסימולציה הדינמית, DynamicNetwork.ned

StatsCollector .4.1

"מוטיבציה" – במערכת קיימות כמוות גדולה (ומשתנה) של ייחידות מכל מודול, וחלק מהסתטיסטיות הנאספות מצריכות עיבוד נוסף (מייצוע, סכמה, חישוב ביחס לסטט' אחרת...). כאשר כל מודול הוציא את הסיגנלים האלו אינדיידואלית, נוצרה לנו כמוות אידירה של מידע שחלקו הגadol לא רלוונטי בפני עצמו, והצריך חישוב נוסף לאחר הסימולציה.

בכדי לפטור את הבעיה הזאת – יצרנו מודול לאיסוף סטטיסטיקה. המודולים שולחים את הנתונים הרלוונטיים (בעזרת קריאה לפונק' public מתוך הקלאס), והוא אוגר את המידע, מבצע את העיבוד הדרוש ומוציא את הסטטיסטיות בצורה מרוכזת, רלוונטית ותמצחית.

הסתטיסטיות שאוספים מתוך קלאס זה : ההשיהה הכלולת (מייצרת הודעה עד לסיום הטיפול בה), מס' העבודות שייצרו, מס' העבודות שהושלמו (ע"י הקטלוגים והמתאים בנפרד), מס' העבודות שנוצרו (dropped), יחס בין כמהות ההודעות שנוצרו/הושלמו.

```

void StatsCollector::initialize() {
  JobLatencyCombined = registerSignal("JobLatency");
  CompletedJobsCombined = registerSignal("CompletedJobs");
  GeneratedJobscombined = registerSignal("GeneratedJobs");
  CompletedOrchWork = registerSignal("CompletedOrchJobs");
  DroppedJobsCombined = registerSignal("DroppedJobs");
  SumCompletedJobs = 0; // initialize at 0 completed jobs at start
  SumGeneratedJobs = 0; // initialize at 0 - generated jobs by Scanners
  SumCompletedOrchWork = 0; // initialize at 0 - orchestrator's done work
  SumDroppedJobs = 0; // initialize at 0 - no dropped jobs when starting
  TotalDroppedVsGenerated = 0; // ratio between dropped jobs to overall generated jobs
  TotalDroppedVsCompleted = 0; // ratio between dropped jobs to overall completed jobs
}

void StatsCollector:: collectJobLatency(simtime_t JobLatency){
  emit(JobLatencyCombined, JobLatency);
}

void StatsCollector::collectCompletedJobs(long JobCount){
  SumCompletedJobs += JobCount;
  emit(CompletedJobsCombined, SumCompletedJobs);
}

```

Ayar 4.4 : מתודת initialize ו-2 מתודות איסוף מידע במרוכז, מתוך StatsCollector.cc

```

void Orchestrator::handleMessage(cMessage *msg){
    // initialize pointers
    sendEventNext = nullptr;
    emit(UpTimeSignal, UnitActive);
    // Differentiate between messages received - Enable/disable unit, Re-check queue for jobs & end of internal job processing
    if (strncmp(msg->getName(), "end processing-", 15) == 0) {
        // Received self-msg for end of processing
        Job* finishedJob = messageQueue.front(); // store finished job into output buffer
        messageQueue.pop();

        numReceivedAndFinished++; // Increment the counter
        collector->collectCompletedOrchWork(1); // increment the overall completed jobs counter
    }
}

```

אייר 4.5 : דוגמה לקריאה למתודה ב-*Orchestrator.cc* ממודול אחר (מתוך StatsCollector)

```

void StatsCollector::finish(){
    TotalDroppedVsGenerated = static_cast<float>(SumDroppedJobs)/static_cast<float>(SumGeneratedJobs);
    TotalDroppedVsCompleted = static_cast<float>(SumDroppedJobs)/static_cast<float>(SumCompletedJobs);
    recordScalar("droppedGeneratedRatio", TotalDroppedVsGenerated);
    recordScalar("droppedCompletedRatio", TotalDroppedVsCompleted);
}

```

אייר 4.6 : מתודות initialize ו-2 מתודות איסוף מידע במרוכז, מתוך StatsCollector.cc

Queue.4.2

הסתטיטיקות שאווסףים בתוך קלאס זה : איבוד הודעות (אם התור מלא), אורך התור, השהייה התור
(latency)

קלאס התור ייחסת פשוט, ועובד באופן הבא :

- **אתחול** : רישום (register) של הסטטיטיקות שנאסוף, שליפה של ערכי הפרמטרים כפי שהוגדרו ב-*ini* / ערך בירית-המחלל (אורך התור – ניתן להגדירו (1-) בשביל תור אינסופי) ויצירת המשק עם *.StatsCollector*

- **בקבלת הודעה** :
 - אם מדובר בהודעה ממודול אחר (כלומר עבודה חדשה שצריכה להיכנס לתור) : בודקים האם התור מלא (אם כן – זורקים את ההודעה, אם לא – מעדכנים בתוך ההודעה את זמן ההגעה לתור, מחזיקים אותה במאפר ושולחים הודעה עצמית המסללת את העיבוד של ההודעה בכניסה לתור – התור יודע לבצע עיבוד על הודעה אחת כל פעם).
 - אם מדובר בהודעה עצמית (כלומר סיימנו עיבוד ראשוני עבור הודעה שהגיעה קודם לכן) : מעבירים את ההודעה/עבודה (המקורית) אל התור.

- **בקשת שליפה מהתור** (מופעל ע"י המודול השולף) : מחשבים את משך השהייה בתור (זמן נוכחי פחות זמן הכניסה לתור), ומעבירים למודול המבקש את ההודעה בראש התור.

בנוסף, הגדרנו פונקציית עוזר לטובת הוצאת הסטטיטיקות הדרושות בשלבים השונים, ופונקציה המנהלת את GUI – מצינה את כמות העבודות בתור.

```

Job* Queue::popJob() {
    // Pop a job from queueToSend and return it
    if (!queueToSend.isEmpty()) {
        EV << getFullName() << ":" << queueToSend.getLength() << endl;
        Job* jb = check_and_cast<Job*>(queueToSend.pop());
        emit(queueLatency, simTime() - jb->getQueue_arrivalTime()); // emits the latency of the queue
        emit(queueLengthSignal, queueToSend.getLength());
        updateDisplay();
        return jb;
    }
    return nullptr; // Return nullptr if the queue is empty
}

```

אייר 4.7 : מתודה *popJob* (נקראת ע"י מודולים אחרים, זו public method, מתוך Queue.cc)

Scanner.4.3

הסתטיסטיות שאווסףים מהתוך קלאס זה : מס' הודעות שנוצרו.

קלאס זה מגדיר את רכיב ה-Scanner, שתפקידם הוא ליצר הודעות למערכת (בקצב שנקבע ע"י הParmeter *(interArrivalTime*

המודול מדמה קצב הגעה פואסוני (נקבע עפ"י הParmeter וניתן להגדלה בכל אופן שנרצה) של הודעות באמצעות שליחת הודעת עצמית בתזמנון הניל. כאשר הוא מקבל הודעה עצמית, הוא מייצר עבודה (job) חדשה ושולח אותה אל התור.

בנוסף, מעדכנים את ה-GUI (כמוות ההודעות שנוצרו עד כה) ואת הסטטיסטיות (ע"י קריאה לפונק' מתוך קלאס ה-StatsCollector) עם קבלת ההודעות.

```
void Scanner::handleMessage(cMessage *msg) {
    if (msg == sendEvent) {
        Job *job = new Job(("msg-" + std::to_string(numJobsCreated)).c_str());
        job->setStartTime(simTime()); // Store the initialization time of the job into it
        send(job, "out", 0);
        numJobsCreated++;
        collector->collectGeneratedJobs(1); // increment overall generated jobs counter
        scheduleAt(simTime() + par("interArrivalTime").doubleValue(), sendEvent); // next msg is in interArrivalTime
        updateDisplay();
    }
}
```

.Scanner.cc ,handleMessage אירור 4.8 : מוגדרת מהתוך

```
void Scanner::updateDisplay() {
    if (hasGUI()) {
        char buf[64];
        sprintf(buf, "Jobs created: %d", numJobsCreated);
        getDisplayString().setTagArg("t", 0, buf); // tag "t" is for text
        getDisplayString().setTagArg("t", 1, "above"); // position above
        getDisplayString().setTagArg("t", 2, "darkblue"); // text color
    }
}
```

.Scanner.cc ,updateDisplay אירור 4.9 : מוגדרת מהתוך

Orchestrator & Catalog .4.4

הסטטיסטיקות שאוספים בתוך קלאסים אלו : זמן פעיל (uptime) וזמן עבודה (idle), כמוות עבודות שטופלו.

הקלאסים המתארים את המתאים (Orchestrators) והקטלוגים (Catalogs) דומים, כאשר השוני ביניהם הוא במשמעותם של תורמים שלופיים ולאילו זוחפים, ובטיפול בעבודות.

המודולים יודעים לקבל את ההודעות הבאות :

- הודעות עצמאיות :

- כאשר סיימנו לעבוד על העבודה/הודעה "הנוכחית". הקטלוג מוציא

בثور סיגナル את משך זמן החיים הכלול של ההודעה ומשמיד אותה (retire), והמתאם מעביר את העבודה אל התור הבא.

- כאשר אנחנו מנסים להשיג עבודה מהטור (ע"י ביצוע polling לטור, שמתבצע לאחר שליחה עצמאית של הودעה כזו עד שמצליכים לשלוּף עבודה מהטור).

- הודעות מיחידת הבדיקה (Controller) :

- הודעה על כיבוי היחידה. בקבלת הודעת סגירה אנחנו מבטלים הודעה

עצמאית polling (אם קיימת), או מסיימים עיבוד שכבר מתבצע, ולא מנסים להשיג עבודה נוספת מהטור.

- הודעה על הפעלת היחידה. במצב זה אנחנו מנסים לשלוּף עבודה מהטור (אם לא קיימת – ממשיכים לבצע polling).

בנוסף, הגדרנו פונקציית GUI מקיפה שמעדכנת את כמות העבודות שהושלמו בכל יחידה ובעת שליפת עבודה מהטור מוצג bubble על כך. בנוסף, למעקב אחרי סטטוס היחידות, אנחנו מנסים את צבע הטקסט בהתאם :

כתום – היחידה דלוכה ונמצאת בעבודה (currently processing), **ירוק** – היחידה דלוכה אך לא נמאת בעבודה (idle), **אדום** – היחידה כבויה (disabled).

```
void Catalog::RequestWork(){  
    Job *jobToProcess = InputqueueModule->popJob(); // Try to get a job from input queue  
  
    if (jobToProcess != nullptr)  
    { // If we got a valid job from queue (if not - nullptr is returned)  
        // create & time a self-message indicating the end of the processing for current job  
        if (hasGUI()) bubble("Pulled job from queue!");  
        sendEventNext = new cMessage(("end processing-" + std::to_string(countOfMessages)).c_str());  
        sendEventNext->setKind(1); // to be able to differentiate between types of self-messages  
        messageQueue.push(jobToProcess);  
        countOfMessages++;  
        scheduleAt(simTime() + par("TimeToConsume").doubleValue(), sendEventNext);  
    }  
    else  
    { // Input queue was empty - no valid jobs received.  
        // Schedule a self-message to re-check queue  
        sendEventNext = new cMessage(("poll queue-" + std::to_string(countOfMessages)).c_str());  
        sendEventNext->setKind(2); // to be able to differentiate between types of self-messages  
        scheduleAt(simTime() + par("TimeToWaitIfQueueIsEmpty").doubleValue(), sendEventNext);  
    }  
}
```

אייר 4.10 : מתודת Catalog.cc , RequestWork

Controller.4.5

הסתטיסטיות שאווסףים בתחום קלאס זה: כמוות הודעות בקרה שנשלחו, כמוות רכיבים דלוקים (מתאים/קטלוגים).

מנגנון הבקרה משתמש לפי פונקציית הבקרה שבה בחרנו לשימוש (נקבע בתחילת הסימולציה ע"י פרמטר controlFuncUsed), כאשר פונקציות הבקרה ניתנות לשילטה מלאה ע"י פרמטרים שונים (סה"כ 18 פרמטרים).

באופן כללי, אופן העבודה הוא:

1. אתחול המשתנים השונים ע"י הפרמטרים שהוגדרו, שליחת הודעות כיבוי למצב ההתחלתי של המערכת (ברירת מחדל – ייחידה אחת מכל סוג דלוקה), שליחת הودעה עצמאית לפחות פעם אחת בה נרצה לדוגום את מצב המערכת.

2. בקבלה הודהה עצמאית:

a. שליפה של אורכי התורים (ע"י פונקציית public `len` לכך בתחום קלאס התור).

b. חישוב של כמוות היחידות הרצוייה מכל סוג (בהתאם לפונק' הבקרה והפרמטרים שנבחרו).

c. שליחת הודעות כיבוי/הדלקה ליחידות הרלוונטיות, שליחת הודהה עצמאית שוב.

קלאס זה כתוב באופן מודולרי לחלוון ומאפשר לשנות בקלות את אופן העבודה של הפונקציות הקיימות (ע"י ערכיים שונים לפרמטרים) או הוספה של פונקציות חדשות עם הפרמטרים הרלוונטיים להן.

```
void Controller::applyScalingCatalogs(int averageSample){  
    // Up-scaling mechanism: linear addition of pods, where maximal deployment is achieved for  
    // queueFullThrottle capacity, and 0 capacity leads to numOfMinOrch.  
    int desired_amount = std::min(  
        Q2_samples[counter % Q2_samples.size()]*((MaxCatalogNum-numOfMinCatalog)/queue2FullThrottle + numOfMinCatalog), // up-scaling linear equation  
        MaxCatalogNum); // lower boundary  
    if (desired_amount > CurrentCatalogAmount){ // if we need more pods than currently enabled  
        EnableCatalogs(CurrentCatalogAmount, desired_amount); // enable the extra pods  
    }  
    // Down-scaling mechanism - has an initial ramp-up time  
    else if (counter >= PodReductionThreshold){ // if the down-scaling is active (has a starting delay)  
        // calculate how many pods are supposed to be deployed based on the last sample's average  
        int avgDesiredAmount = averageSample*(MaxCatalogNum-numOfMinCatalog)/queue2FullThrottle + numOfMinCatalog;  
        // Down-scaling mechanism: if the # of pods using the average is lower than current deployed number and >= amount suggested by latest sample  
        if (avgDesiredAmount < CurrentCatalogAmount && avgDesiredAmount >= desired_amount){  
            EV << "Too many Catalogs active, Down-scaling from " << CurrentCatalogAmount << " to " << avgDesiredAmount << endl;  
            DisableCatalogs(avgDesiredAmount, CurrentCatalogAmount); // disable the extra pods  
        }  
    }  
}
```

אייר 4.11 : מתודות Controller.cc (חלק מפונק' הבקרה הליינארית).

```

simple Controller {
    parameters:
        @signal[orchPodsNum](type="long");
        @signal[catalogPodsNum](type="long");
        @signal[ControlMessagesSent](type="long");
        @statistic[ControlMessagesSent](title="number of control messages sent"; record=vector,stats,count;);
        @statistic[catalogPodsNum](title = "Number of catalog pods"; record=vector,mean,max;);
        @statistic[orchPodsNum](title="Number of orch pods"; record=vector,mean,max;);

    // parameter declarations
    // permanent parameters
    int numOrchs = default(12); // 11 is the needed amount for queue stability
    int numcatalogs = default(175); // slightly above 171 orchs for stable queue
    int numMinOrchs = default(3); // lowest number of orchs possible
    int numMinCatalog = default(6); // lowest number of catalogs possible
    string controlFuncUsed = default("Linear");
    double SamplingFreq = default(0.5); // sample every 0.5 seconds

    // Linear control function related
    int queue1SamplingHistory = default(1200); // 10 minutes' history
    int queue2SamplingHistory = default(1200); // 10 minutes' history
    int queueFullThrottle = default(100); // when queue reaches it, all pods are deployed
    int PodReductionThreshold = default(240); // down-scaling possible only after 2 minutes

    int samplingPairIndex = default(1); // For zipping elements in the .ini file (parameter sweep)

    // QDTE control function related
    double orchServiceRate = default(0.25);
    double catalogServiceRate = default(5);
    int stabilityThreshold = default(5);
    double latencyTargetQueue1 = default(0.01); // 10x the avg for static sim
    double latencyTargetQueue2 = default(0.08); // 10x the avg for static sim
    int increaseRateQueue1 = default(1);
    int increaseRateQueue2 = default(1);

    gates:
        output orch_out[];
        output catalog_out[];
}

```

איור 4.12 : הגדרות NED עבור קלאס Controller.

Omnetpp.ini File .4.6

הגדרות הסימולציה השונות בינויו בצורה היררכית, כאשר לכל סוג סימולציה יש config בסיסי משלו, וממנו configs שיורשים ומוסיפים הגדרות שונות עפ"י הצורך (parameter sweeps, scenarios testings, ..override settings).

```

[Config QDTE_SamplingRate_1_00]
extends = QDTE_Sweep_base
experiment-label = ${repetition} ++ ${configname}
repeat=5

# Controller-specific configurations - Linear function
*.controller.numMinOrchs = 1
*.controller.numMinCatalog = 1
*.controller.SamplingFreq = 1
*.controller.stabilityThreshold = ${5,15,30,60}

*.controller.samplingPairIndex = ${0..2} # used here for zipping together latency targets
*.controller.latencyTargetQueue1 = ${select(samplingPairIndex, 0.01, 0.005, 0.002)} # compared to static: x10, x5, x2
*.controller.latencyTargetQueue2 = ${select(samplingPairIndex, 0.08, 0.04, 0.016)} # compared to static: x10, x5, x2

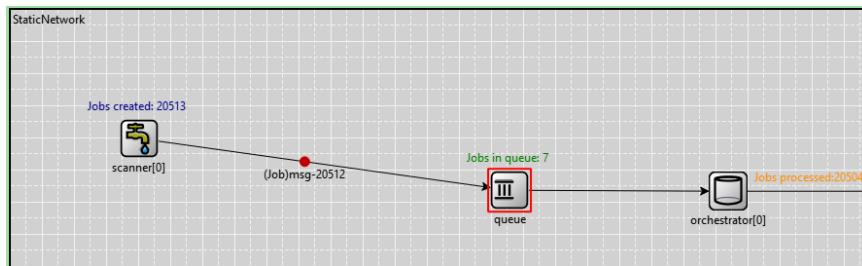
```

איור 4.13 : דוגמה ל config עבור תרחיש בסימולציה הדינמית (parameter sweep with QDTE controller)

5. בדיקת שפיפות

5.1 התאמה בין הערכות התיאורטיים להתוצאות הסימולציה

בכדי לוודא שהסימולציה שלנו עובדת בצורה תקינה ומציאותית, בדgesch על התורים, נרים בדיקה על התור הראשון, כאשר הוא מזון-m Scanner אחד ומזין Orchesterator יחיד, שניהם ממודלים בתור תהליכים אקספוננציאליים עם קבועים של 0.3 ו-0.25 בהתאמה (הקצב האמיטי של ה-Scanner הוא 0.1 ולא 0.3, אך החישובים התיאורטיים נכונים רק עבור תורים יציבים, לכן לשם הבדיקה הורדנו את קצב הכניסות).



אייר 5.1 : GUI סימולציה בדיקת השפיפות. ניתן לראות הוודה בדרך אל התור והרצן (Orchestrator) נמצא בעבודה.

בהתאם לתיאוריה, נצפה לקבל:

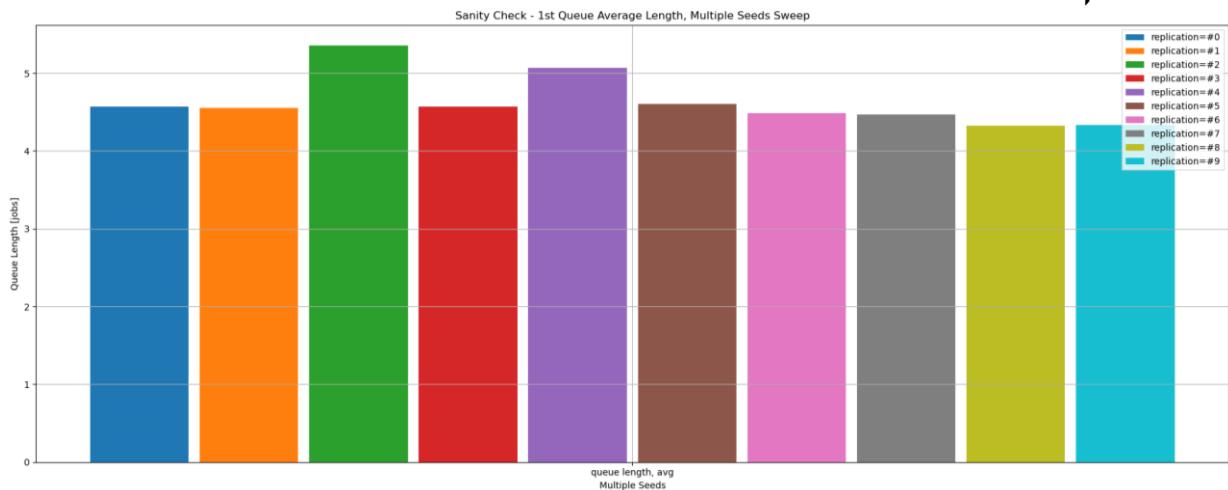
$$\lambda = \frac{1}{arrival\ rate} = \frac{1}{0.3} = 3.33 ; \mu = \frac{1}{service\ rate} = \frac{1}{0.25} = 4$$

$$E[Queue\ length] = \frac{\lambda^2}{\mu(\mu - \lambda)} = \frac{25}{6} = 4.16$$

$$E[Time\ in\ queue] = Latency = \frac{\lambda}{\mu(\mu - \lambda)} = 1.25$$

הריצו את הסימולציה 10 פעמים (10 סידים שונים), וקיבלו את התוצאות הבאות:

5.1.1 אורץ תור:



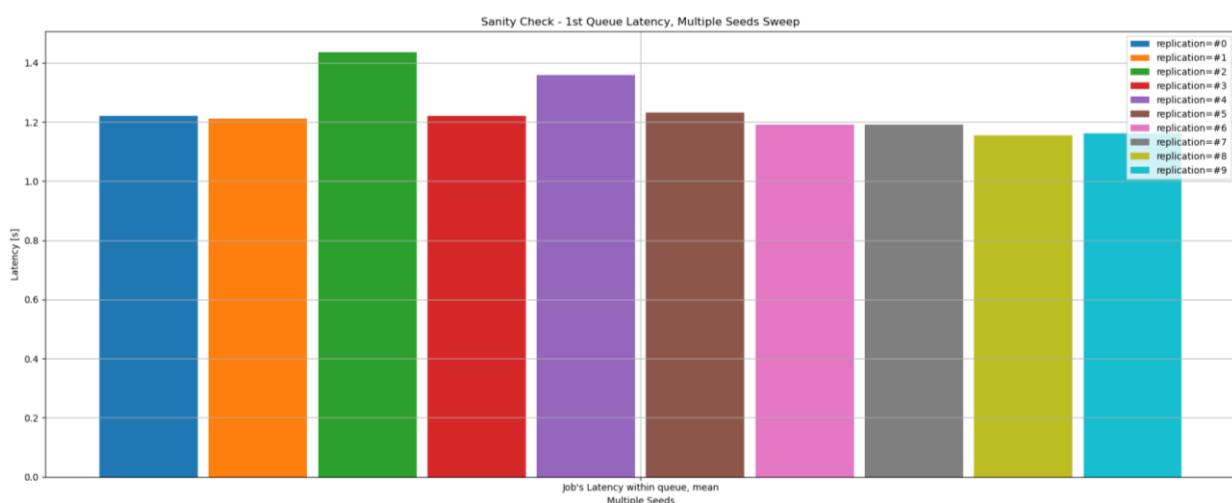
אייר 5.2 : אורץ התור הממוצע עבור 10 הרצות עיג סידים שונים.

Experiment	Measurement	Replication	Module	Name	Count	Mean	StdDev
sanityCheck		#0	StaticNetwork.queue	queueLength:vector	67 003	~ 4.57191	~ 5.10315
sanityCheck		#1	StaticNetwork.queue	queueLength:vector	67 080	~ 4.55242	~ 5.03444
sanityCheck		#2	StaticNetwork.queue	queueLength:vector	66 708	~ 4.5312	~ 6.70328
sanityCheck		#3	StaticNetwork.queue	queueLength:vector	66 630	~ 4.56898	~ 4.83379
sanityCheck		#4	StaticNetwork.queue	queueLength:vector	66 336	~ 5.06886	~ 5.95971
sanityCheck		#5	StaticNetwork.queue	queueLength:vector	66 919	~ 4.60055	~ 4.97043
sanityCheck		#6	StaticNetwork.queue	queueLength:vector	66 997	~ 4.48395	~ 4.99503
sanityCheck		#7	StaticNetwork.queue	queueLength:vector	67 086	~ 4.46527	~ 4.70169
sanityCheck		#8	StaticNetwork.queue	queueLength:vector	66 664	~ 4.32551	~ 4.8363
sanityCheck		#9	StaticNetwork.queue	queueLength:vector	66 492	~ 4.329	~ 4.82571

טבלה 5.1 : סטטיסטיות אורך התור עבור 10 הרצות ע"ג סידים שונים.

ניתן לראות כי האורך המתקבל ממוצע (4.73) קרוב לתוצאה התיאורית. ההפרש צפוי בגלל מגנון העיבוד שהבכנת החבילה לתור, שמוסיף בממוצע ~1ms ומשפיע על התוצאה המתתקבלת בפועל.

5.1.2. השהייה/זמן מצטבר בתור:



אייר 5.3 : אורך התור הממוצע עבור 10 הרצות ע"ג סידים שונים.

Experiment	Measurement	Replication	Module	Name	Count	Mean	StdDev
sanityCheck		#0	StaticNetwork.queue	queueLatency:vector	33 499	~ 1.2217s	~ 1.40925s
sanityCheck		#1	StaticNetwork.queue	queueLatency:vector	33 540	~ 1.21194	~ 1.38025
sanityCheck		#2	StaticNetwork.queue	queueLatency:vector	33 354	~ 1.43484	~ 1.83205
sanityCheck		#3	StaticNetwork.queue	queueLatency:vector	33 315	~ 1.22175	~ 1.33371
sanityCheck		#4	StaticNetwork.queue	queueLatency:vector	33 168	~ 1.35863	~ 1.62459
sanityCheck		#5	StaticNetwork.queue	queueLatency:vector	33 453	~ 1.23129	~ 1.35899
sanityCheck		#6	StaticNetwork.queue	queueLatency:vector	33 492	~ 1.19176	~ 1.38063
sanityCheck		#7	StaticNetwork.queue	queueLatency:vector	33 543	~ 1.19109	~ 1.29011
sanityCheck		#8	StaticNetwork.queue	queueLatency:vector	33 332	~ 1.15595	~ 1.33357
sanityCheck		#9	StaticNetwork.queue	queueLatency:vector	33 246	~ 1.16108	~ 1.35592

טבלה 5.2 : סטטיסטיות ההשהייה בתור עבור 10 הרצות ע"ג סידים שונים.

קיבלו השהייה ממוצעת של 1.238, קרובה מאוד לערך התיאורי (1.25).

לסיכום, הערכים שהתקבלו בסימולציה תואמים את הערכים התיאורתיים.

6. תוצאות – סימולציה סטטיסטית

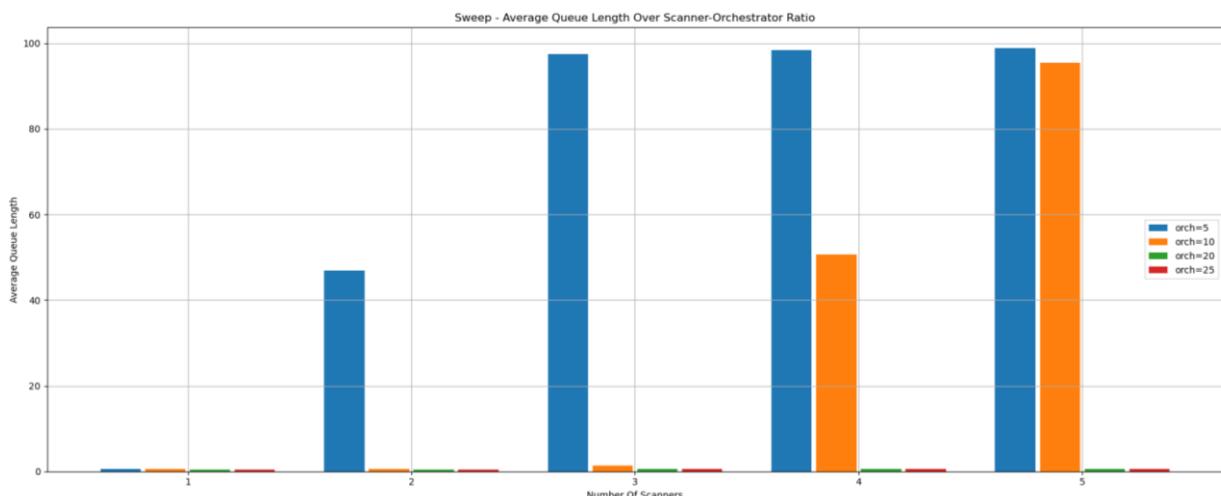
6.1. סימולציה סטטיסטית: מבחן עבודה רגיל – 3 Scanners

6.1.1. איזון התווך הראשוני (Scanners-Orchestrators)

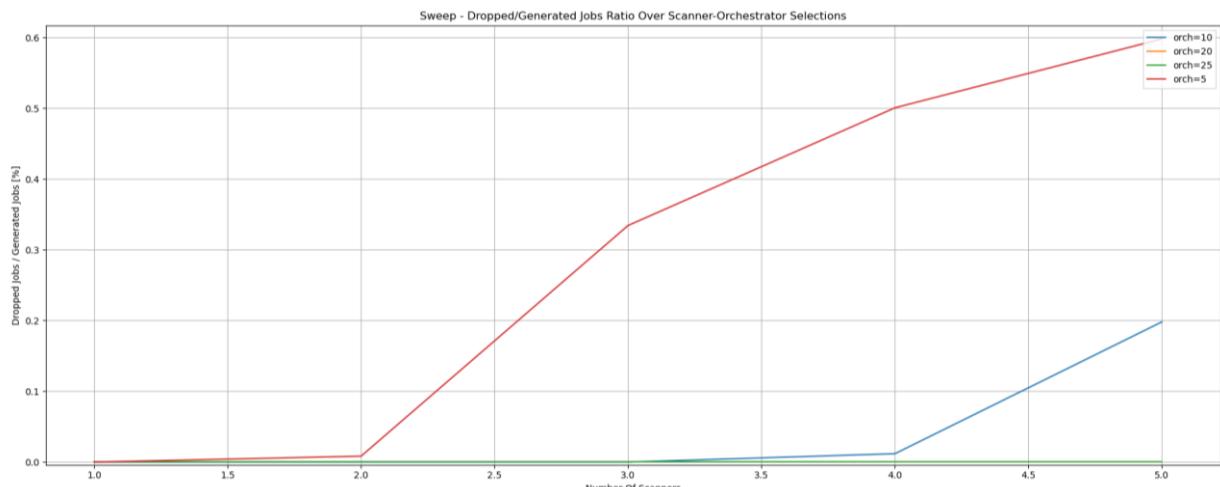
נרצה לאוזן תחילת את התווך הראשוני בסימולציה שלנו. בכך לבטל את ההשפעה של כמות הקטלוגים והתווך השני נגדיר אותו להיות לא-חסום עבור חלק זה של הבדיקות. שינוי זה מאפשר לנו להגדיר רק קטלוג יחיד בשלב זה של הסימולציה (בשביל שהסימולציה מפעילה), ובכך להפחית את כוח החישוב הדרוש כאשר אין בו צורך.

6.1.1.1. סריקה ראשונית כוללת

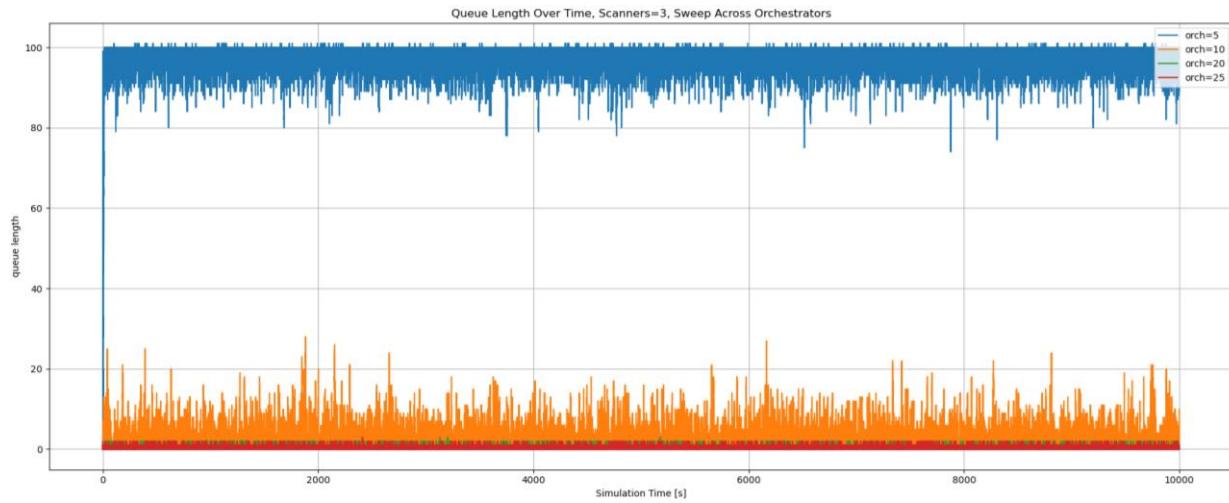
מבצע Sweep על כמות ה-Orchestrator-Scanner נקודת העבודה שלנו (נבדוק בתחום 1-5) כדי למצוא את התוצאות אותן נרצה לבדוק לעומק. נבדוק עבור כמות היחידות הבאות: 5,10,20,25.



אייר 6.1 : גרפ אוורך התווך הממוצע כפונקציה של כמות ה-Scanners, לערכי Orchestrators .5,10,20,25

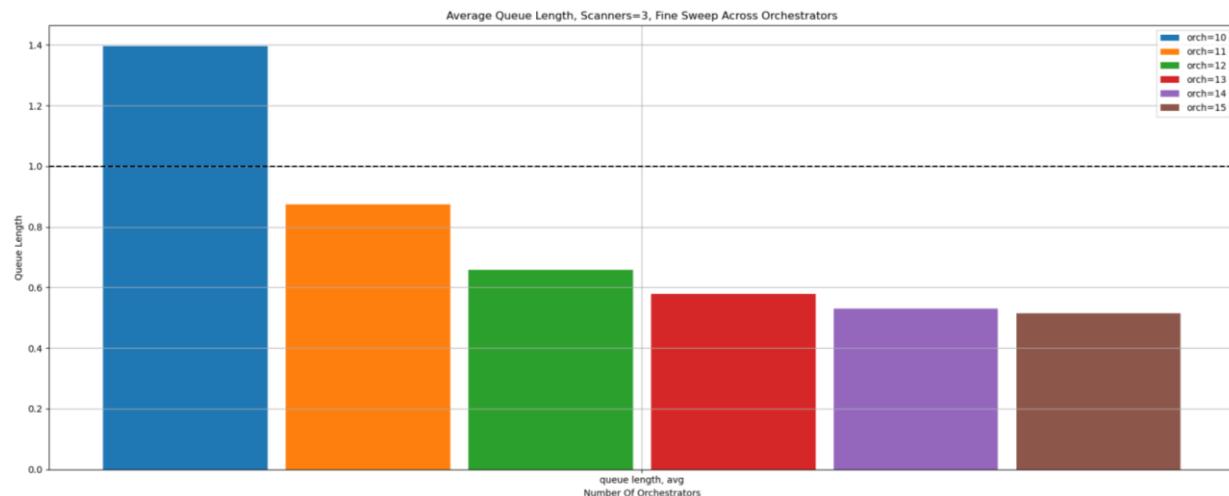


אייר 6.2 : גרפ של היחס בין איבודי העבודות (dropped jobs) לכמות העבודות הכוללת, כפונקציה של כמות ה-Scanners, לערכי Orchestrators .5,10,20,25

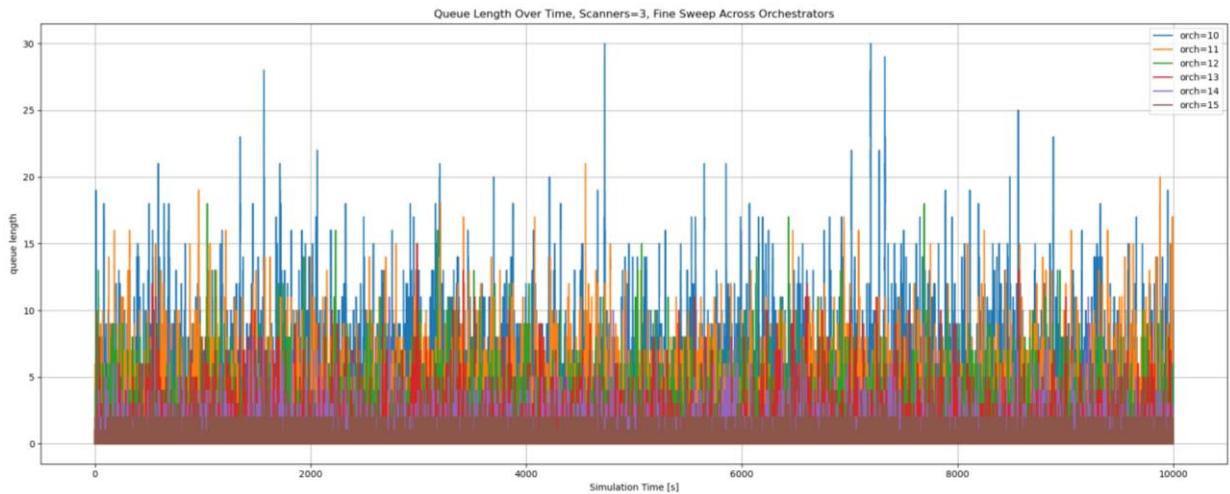


אייר 6.3 : אורך התור לאורך זמן עבור 3 סורקים. ניתן לראות כי עבור 5 מתאימים התור מגיע לקיבולת שלו (100).

הדרישה שלנו היא שיהיו 0 איבודי עבודות, מה שמתאפשר עבור $Orchestrators \geq 10$. בנוספ', ככל לשיטת לב כי אורך התור הממוצע הינו קטן מ-1 בתחום שבין 10-20 מתאימים (Orchestrators), כאשר כבר עבור 10 מתאימים אנו קרובים למדייל-1. כיוון שהוא מצב העבודה המקורי בו המערכת צפוייה לשחות לאורך זמן רב מאוד, חשוב לנו להקפיד על כך שאורך התור הממוצע יהיה קטן מ-1, בשביל להבטיח שהטור לא יתפוץ לאורך זמן. על כן – נתמקד בתחום של 10-15 מתאימים ונתמקד עבור 3 סורקים :



אייר 6.4 : אורך התור הממוצע כפונקציה של כמות המתאימים. הקו המקווקו מסמן אורך תור ממוצע 1.



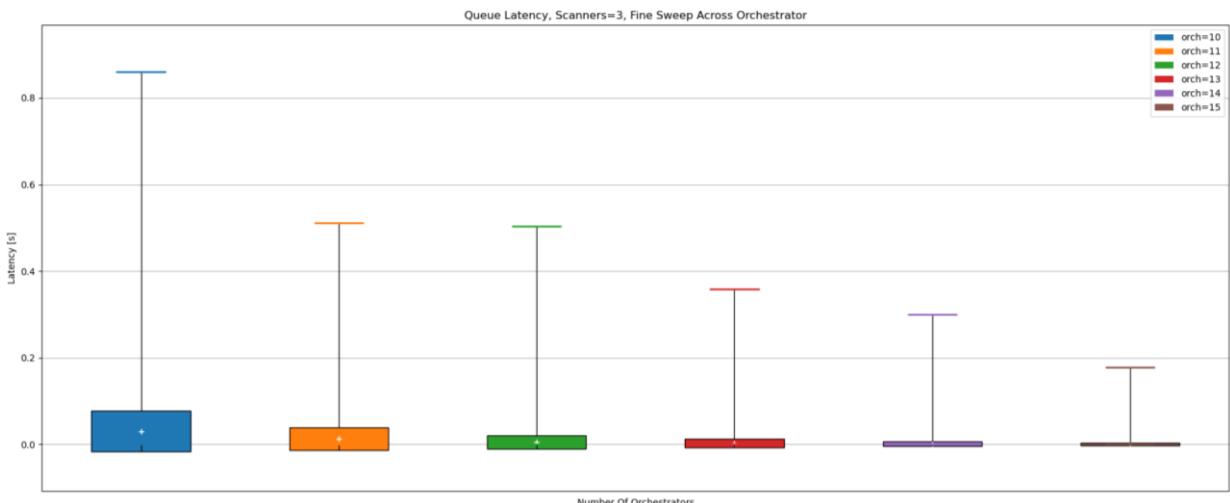
אייר 6.5 : אורך התור לאורץ זמן עברו 3 סורקים ו-10-15 מתאימים.

נוכל לראות גם כי אף אחד מן האפשרויות שנבדקו לא מוביל לאיבוד עבודות :

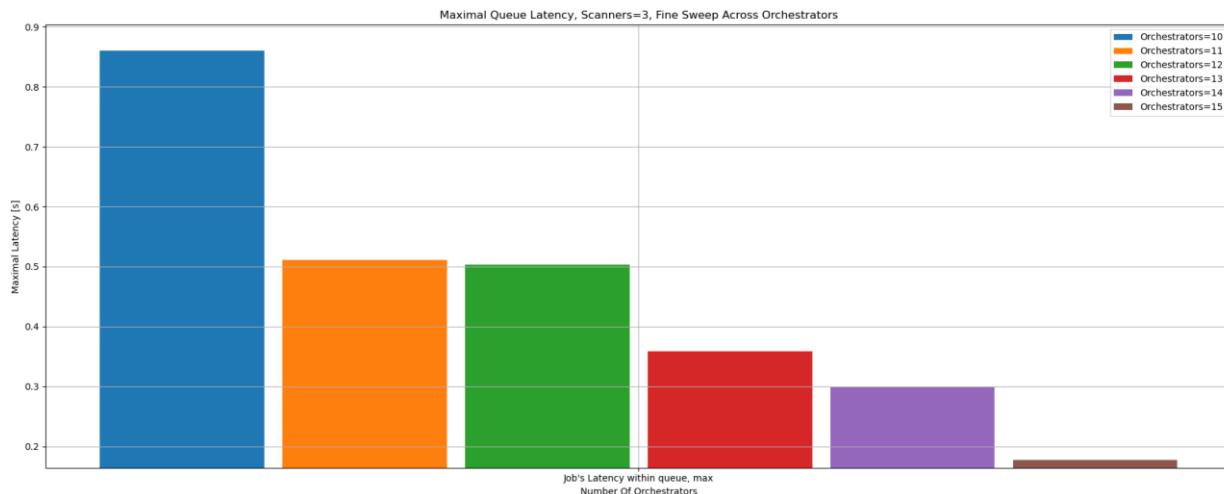
Experiment	Measurement ^	Replication	Module	Name	Value
scanner_Orchs_Sweep_queue_100	\$catalog=1, \$scan=3, \$orch=10	#0	StaticNetwork.statsCollector	droppedGeneratedRatio	0
scanner_Orchs_Sweep_queue_100	\$catalog=1, \$scan=3, \$orch=11	#0	StaticNetwork.statsCollector	droppedGeneratedRatio	0
scanner_Orchs_Sweep_queue_100	\$catalog=1, \$scan=3, \$orch=12	#0	StaticNetwork.statsCollector	droppedGeneratedRatio	0
scanner_Orchs_Sweep_queue_100	\$catalog=1, \$scan=3, \$orch=13	#0	StaticNetwork.statsCollector	droppedGeneratedRatio	0
scanner_Orchs_Sweep_queue_100	\$catalog=1, \$scan=3, \$orch=14	#0	StaticNetwork.statsCollector	droppedGeneratedRatio	0
scanner_Orchs_Sweep_queue_100	\$catalog=1, \$scan=3, \$orch=15	#0	StaticNetwork.statsCollector	droppedGeneratedRatio	0

טבלה 6.1 : כמות העבודות שנאבדו ביחס לכמות העבודות שיוצרו במהלך הסימולציות עבור 10-15 מתאימים.

נסתכל גם על ההשԽיות הממוצעת והמקסימלית עבור העבודות בתוך התור – ככלمر, משך הזמן שעובר מרגע הכניסה לתור עד שהן יוצאות ממנו.



אייר 6.6 : ההשԽה הממוצעת וסטטיות התקן בתור עבור 10-15 מתאימים.



איור 6.7 : ההשיה המקסימלית בתור עבור 10-15 מתאים.

Experiment	Measurement	Replication	Module	Name	Value
scanner_Orchs_Sweep_queue_100	\$catalog=1, \$scan=3, \$orch=10	#0	StaticNetwork.queue	queueLatency:max	~ 860.743 ms
scanner_Orchs_Sweep_queue_100	\$catalog=1, \$scan=3, \$orch=11	#0	StaticNetwork.queue	queueLatency:max	~ 510.761 ms
scanner_Orchs_Sweep_queue_100	\$catalog=1, \$scan=3, \$orch=12	#0	StaticNetwork.queue	queueLatency:max	~ 503.677 ms
scanner_Orchs_Sweep_queue_100	\$catalog=1, \$scan=3, \$orch=13	#0	StaticNetwork.queue	queueLatency:max	~ 358.521 ms
scanner_Orchs_Sweep_queue_100	\$catalog=1, \$scan=3, \$orch=14	#0	StaticNetwork.queue	queueLatency:max	~ 299.699 ms
scanner_Orchs_Sweep_queue_100	\$catalog=1, \$scan=3, \$orch=15	#0	StaticNetwork.queue	queueLatency:max	~ 178.047 ms
scanner_Orchs_Sweep_queue_100	\$catalog=1, \$scan=3, \$orch=10	#0	StaticNetwork.queue	queueLatency:mean	~ 30.2896 ms
scanner_Orchs_Sweep_queue_100	\$catalog=1, \$scan=3, \$orch=11	#0	StaticNetwork.queue	queueLatency:mean	~ 12.5772 ms
scanner_Orchs_Sweep_queue_100	\$catalog=1, \$scan=3, \$orch=12	#0	StaticNetwork.queue	queueLatency:mean	~ 5.46847 ms
scanner_Orchs_Sweep_queue_100	\$catalog=1, \$scan=3, \$orch=13	#0	StaticNetwork.queue	queueLatency:mean	~ 2.69035 ms
scanner_Orchs_Sweep_queue_100	\$catalog=1, \$scan=3, \$orch=14	#0	StaticNetwork.queue	queueLatency:mean	~ 1.12523 ms
scanner_Orchs_Sweep_queue_100	\$catalog=1, \$scan=3, \$orch=15	#0	StaticNetwork.queue	queueLatency:mean	~ 607.976 us

טבלה 6.2 : ההשיה הממוצעת והמקסימלית בתור עבור 10-15 מתאים.

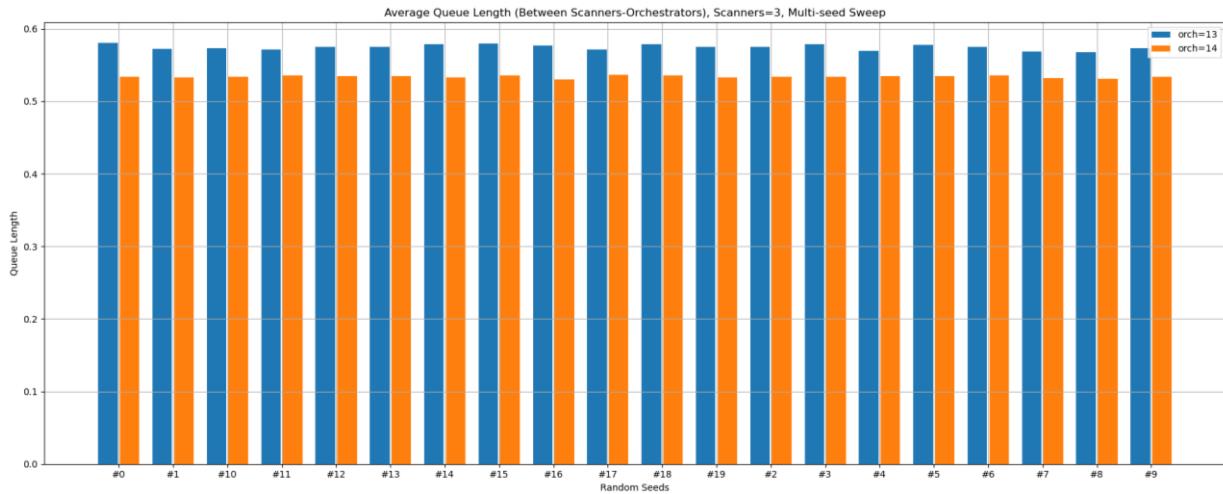
על אף שכבר עבור 11 מתאים התור יהיה יציב לאורך זמן, שימוש ב-13 מתאים במקום 11 מקטין את ההשיה הממוצעת מ- ms 2.690 ל- ms 12.577 (פי 4.7!), ואת ההשיה המקסימלית מ- ms 510.761 ל- ms 358.521 (פי 1.4). זהו שיפור ממשוני, על כן למרות שהטור יציב גם עבור 11 מתאים, הבחירה ב-13 מניבה שיפור ממשוני בBITS.

בנוסף, עבור 14 מתאים ההשיה הממוצעת יורדת ל- ms 1.125 (פי 2.4 מ-13 מתאים, פי 2 מ-11.2 מ-11 מתאים) ואת ההשיה המקסימלית ל- ms 299.699 (שיעור של ms ~58 לעומת ms ~13 מתאים).

OMNet++ מימוש את כל הפעולות הרנדומליות שלו באמצעות seeds (מנגנון pseudo-random), לכן על מנת לבטל את ההשפעה של ה-seed הספציפי שבאמצעותו בדקנו עד כה, נבצע בדיקה מקיפה עם מספר seeds שתזודא את ההתנהגות שראינו עבור 13 ו-14 מתאים, ובהתאם נבחר את הכמות שבאופן עקבי טובה יותר ביחס לעלות (של הוספת רכיב נוסף).

6.1.1.2. בדיקה מקיפה עם מספר seeds

הריצנו את הסימולציה עבור 11 ו-12 מתאים על גבי 20 seeds שונים :



אייר 6.8 : אורך התווך הממוצע בהרצאות השונות עבור 13 ו-14 מטאמים.

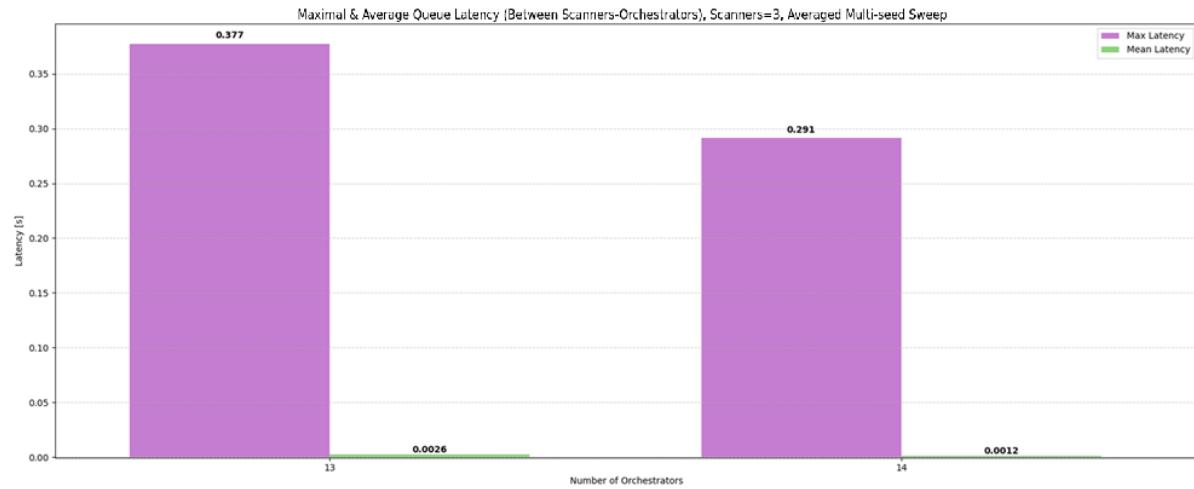
נוכל לראות שהאורך הממוצע עבור 13 מטאמים הוא ~0.57, ועבור 14 מטאמים הוא ~0.53, ככלומר נשאר היבט מתחת ל-1.

בכל הסימולציות, לא היו איבודי עבודות :

Experiment	Measurement	Replication	Module	Name	Value
Scanner_Orchs_Sweep_multi_seed	Catalog=1, \$scan=3, \$orch=13	#0	StaticNetwork.queue	dropped:count	0
Scanner_Orchs_Sweep_multi_seed	Catalog=1, \$scan=3, \$orch=13	#1	StaticNetwork.queue	dropped:count	0
Scanner_Orchs_Sweep_multi_seed	Catalog=1, \$scan=3, \$orch=13	#2	StaticNetwork.queue	dropped:count	0
Scanner_Orchs_Sweep_multi_seed	Catalog=1, \$scan=3, \$orch=13	#3	StaticNetwork.queue	dropped:count	0
Scanner_Orchs_Sweep_multi_seed	Catalog=1, \$scan=3, \$orch=13	#4	StaticNetwork.queue	dropped:count	0
Scanner_Orchs_Sweep_multi_seed	Catalog=1, \$scan=3, \$orch=13	#5	StaticNetwork.queue	dropped:count	0
Scanner_Orchs_Sweep_multi_seed	Catalog=1, \$scan=3, \$orch=13	#6	StaticNetwork.queue	dropped:count	0
Scanner_Orchs_Sweep_multi_seed	Catalog=1, \$scan=3, \$orch=13	#7	StaticNetwork.queue	dropped:count	0
Scanner_Orchs_Sweep_multi_seed	Catalog=1, \$scan=3, \$orch=13	#8	StaticNetwork.queue	dropped:count	0
Scanner_Orchs_Sweep_multi_seed	Catalog=1, \$scan=3, \$orch=13	#9	StaticNetwork.queue	dropped:count	0
Scanner_Orchs_Sweep_multi_seed	Catalog=1, \$scan=3, \$orch=13	#10	StaticNetwork.queue	dropped:count	0
Scanner_Orchs_Sweep_multi_seed	Catalog=1, \$scan=3, \$orch=13	#11	StaticNetwork.queue	dropped:count	0
Scanner_Orchs_Sweep_multi_seed	Catalog=1, \$scan=3, \$orch=13	#12	StaticNetwork.queue	dropped:count	0
Scanner_Orchs_Sweep_multi_seed	Catalog=1, \$scan=3, \$orch=13	#13	StaticNetwork.queue	dropped:count	0
Scanner_Orchs_Sweep_multi_seed	Catalog=1, \$scan=3, \$orch=13	#14	StaticNetwork.queue	dropped:count	0
Scanner_Orchs_Sweep_multi_seed	Catalog=1, \$scan=3, \$orch=13	#15	StaticNetwork.queue	dropped:count	0
Scanner_Orchs_Sweep_multi_seed	Catalog=1, \$scan=3, \$orch=13	#16	StaticNetwork.queue	dropped:count	0
Scanner_Orchs_Sweep_multi_seed	Catalog=1, \$scan=3, \$orch=13	#17	StaticNetwork.queue	dropped:count	0
Scanner_Orchs_Sweep_multi_seed	Catalog=1, \$scan=3, \$orch=13	#18	StaticNetwork.queue	dropped:count	0
Scanner_Orchs_Sweep_multi_seed	Catalog=1, \$scan=3, \$orch=13	#19	StaticNetwork.queue	dropped:count	0
Scanner_Orchs_Sweep_multi_seed	Catalog=1, \$scan=3, \$orch=14	#0	StaticNetwork.queue	dropped:count	0
Scanner_Orchs_Sweep_multi_seed	Catalog=1, \$scan=3, \$orch=14	#1	StaticNetwork.queue	dropped:count	0
Scanner_Orchs_Sweep_multi_seed	Catalog=1, \$scan=3, \$orch=14	#2	StaticNetwork.queue	dropped:count	0
Scanner_Orchs_Sweep_multi_seed	Catalog=1, \$scan=3, \$orch=14	#3	StaticNetwork.queue	dropped:count	0
Scanner_Orchs_Sweep_multi_seed	Catalog=1, \$scan=3, \$orch=14	#4	StaticNetwork.queue	dropped:count	0
Scanner_Orchs_Sweep_multi_seed	Catalog=1, \$scan=3, \$orch=14	#5	StaticNetwork.queue	dropped:count	0
Scanner_Orchs_Sweep_multi_seed	Catalog=1, \$scan=3, \$orch=14	#6	StaticNetwork.queue	dropped:count	0
Scanner_Orchs_Sweep_multi_seed	Catalog=1, \$scan=3, \$orch=14	#7	StaticNetwork.queue	dropped:count	0
Scanner_Orchs_Sweep_multi_seed	Catalog=1, \$scan=3, \$orch=14	#8	StaticNetwork.queue	dropped:count	0
Scanner_Orchs_Sweep_multi_seed	Catalog=1, \$scan=3, \$orch=14	#9	StaticNetwork.queue	dropped:count	0
Scanner_Orchs_Sweep_multi_seed	Catalog=1, \$scan=3, \$orch=14	#10	StaticNetwork.queue	dropped:count	0
Scanner_Orchs_Sweep_multi_seed	Catalog=1, \$scan=3, \$orch=14	#11	StaticNetwork.queue	dropped:count	0
Scanner_Orchs_Sweep_multi_seed	Catalog=1, \$scan=3, \$orch=14	#12	StaticNetwork.queue	dropped:count	0
Scanner_Orchs_Sweep_multi_seed	Catalog=1, \$scan=3, \$orch=14	#13	StaticNetwork.queue	dropped:count	0
Scanner_Orchs_Sweep_multi_seed	Catalog=1, \$scan=3, \$orch=14	#14	StaticNetwork.queue	dropped:count	0
Scanner_Orchs_Sweep_multi_seed	Catalog=1, \$scan=3, \$orch=14	#15	StaticNetwork.queue	dropped:count	0
Scanner_Orchs_Sweep_multi_seed	Catalog=1, \$scan=3, \$orch=14	#16	StaticNetwork.queue	dropped:count	0
Scanner_Orchs_Sweep_multi_seed	Catalog=1, \$scan=3, \$orch=14	#17	StaticNetwork.queue	dropped:count	0
Scanner_Orchs_Sweep_multi_seed	Catalog=1, \$scan=3, \$orch=14	#18	StaticNetwork.queue	dropped:count	0
Scanner_Orchs_Sweep_multi_seed	Catalog=1, \$scan=3, \$orch=14	#19	StaticNetwork.queue	dropped:count	0

טבלה 6.3 : כמות העבודות שנאבדו במהלך הסימולציות עבור 13 ו-14 מטאמים (ע"ג seeds שונים).

נסתכל על ההשיהה (המקסימלית והממוצעת) בין 2 האפשרויות – נמצוא את התוצאות על פני 20 הרצאות :



אייר 6.9 : ההשיה הממוצעת והמקסימלית בטור עbor 13 ו-14 מתאימים (ממוצע עיג seeds שונים).

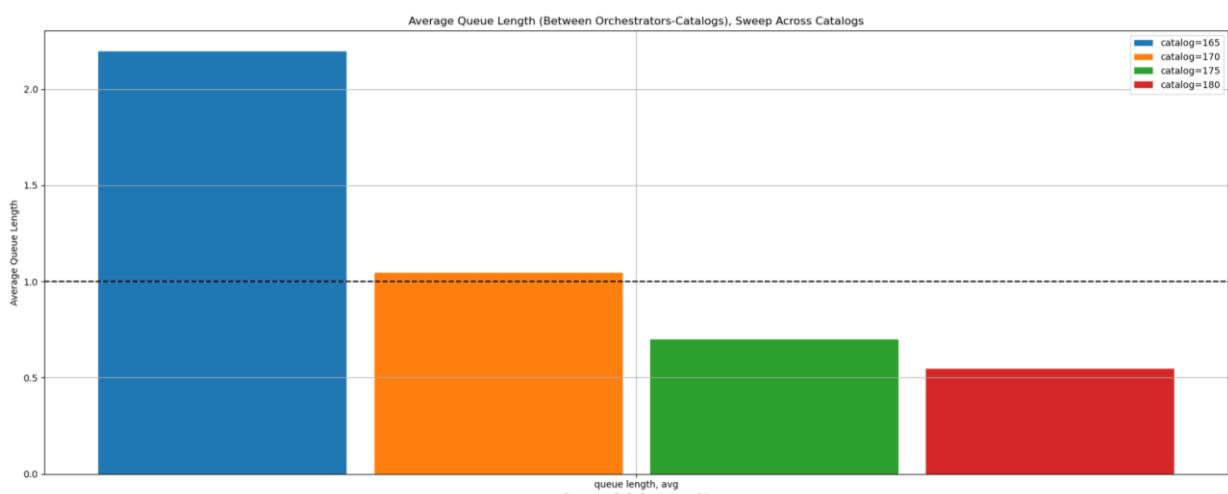
יש שיפור של 30% בהשיה המקסימלית ו-216% בהשיה הממוצעת – עbor מחיר של הוספה יחידה אחת זהו שיפור משמעותי, על כן נבחר להשתמש במצב הסטטי ב-14 מתאימים.

6.1.2. איזון התור השני (Orchestrators-Catalogs)

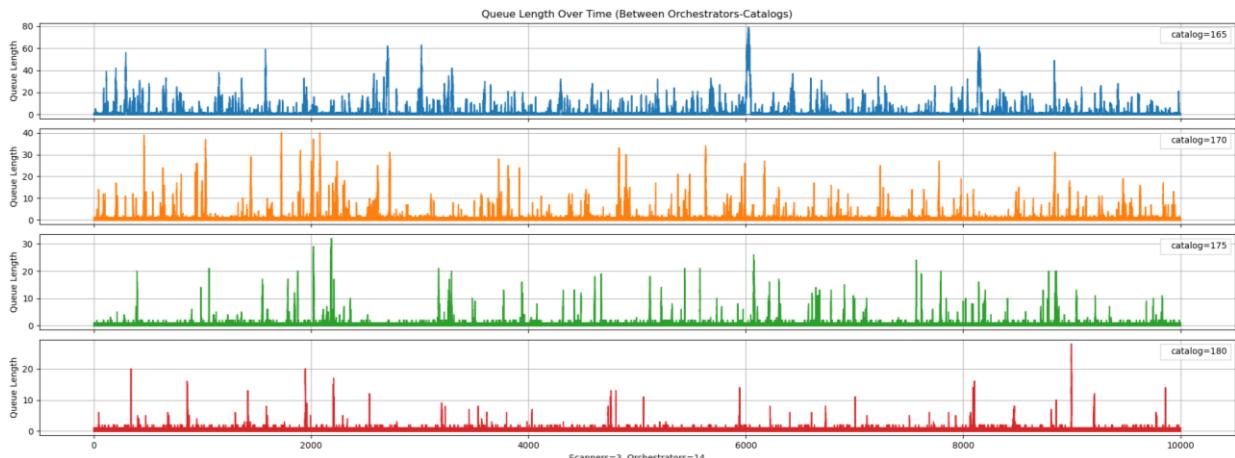
כעת נחזר את מגבלת האורך לתור השני (100), נשתמש בכמה המתאימים שמצאנו קודם (14) ונאזן את התור השני.

6.1.2.1. סריקה ראשונית

בוצע סריקה ראשונית למציאת תחום כמות ייחidot הקטלוגים שמעניין אותנו. נבדק עbor 180-165 :

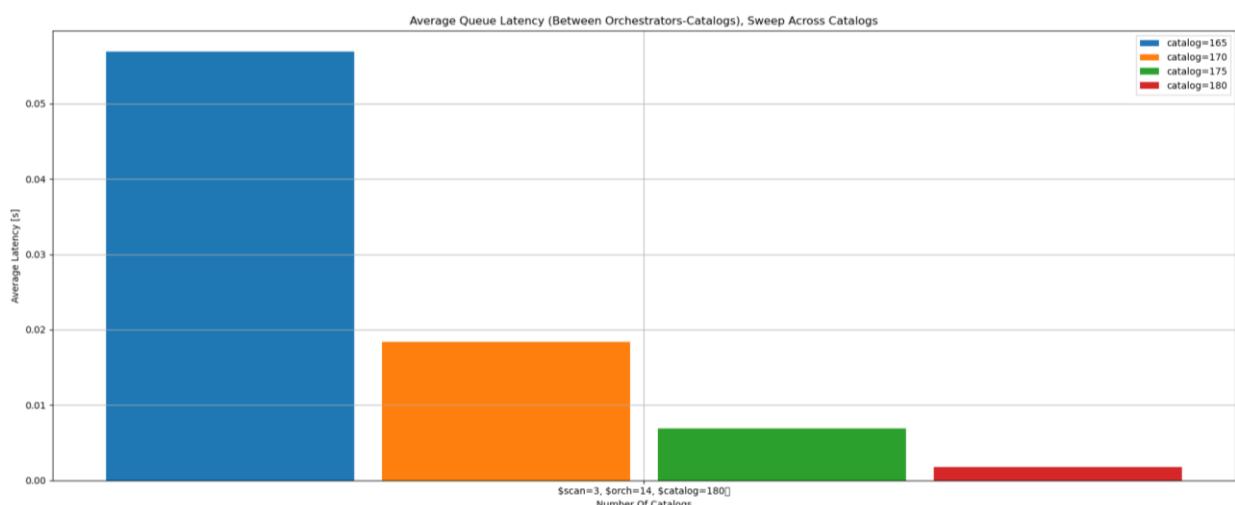


אייר 6.10 : אורך התור הממוצע כפונקציה של כמות המתאימים. הקוו המקווקו מסמן אורך תור ממוצע 1.



אייר 6.11: ארכוי התורים לאורך זמן עבור 165-180 קטלוגים.

עבור 165 קטלוגים, אנחנו מקבלים "פיקים" גבוהים למדדי, שמגיעים עד ל-80 הודעות בתור וקרובים לקיבולת המקסימלית. ככל שמעלים את כמות הקטלוגים, גביהי ה"פיקים" יורדים בכ-20-10~, כאשר הירידה המשמעותית ביותר מתרחשת בין 165 ל-170 קטלוגים – פי 2. נציין שעבור כל האפשרויות אין איובוד עבודות (אורך התור לא מגע ל-100 בשום שלב).

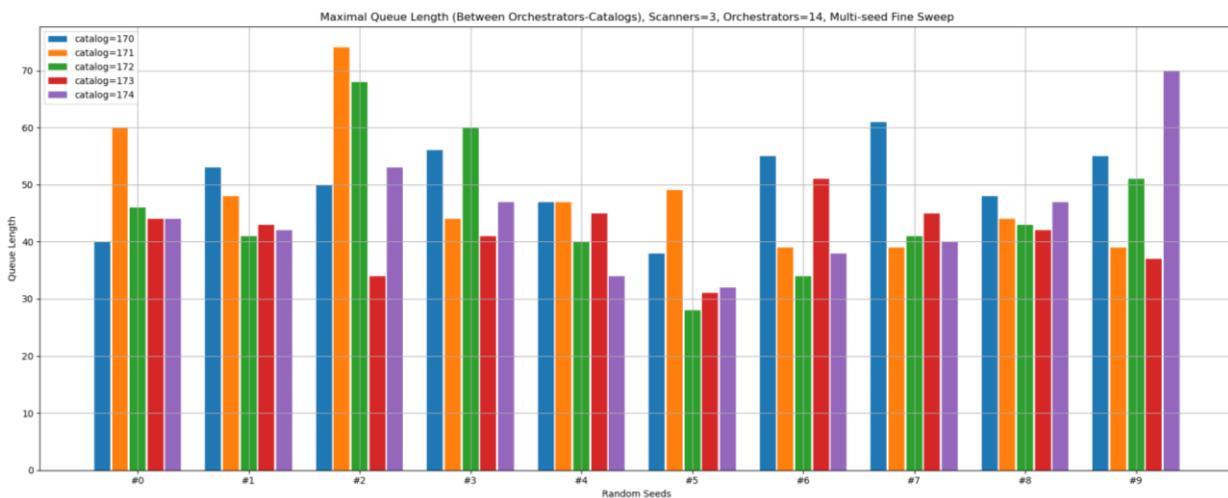


אייר 6.12 : השהיה ממוצעת בתור עבור 165-180 קטלוגים.

במקביל, ההשאה הממוצעת (latency) בתור יורדת בצורה החדה ביותר בין 165 ל-170 יחידות, והוספה של עוד 5 יחידות (ל-175) חותכת את ההשאה הממוצעת פי יותר מ-2.

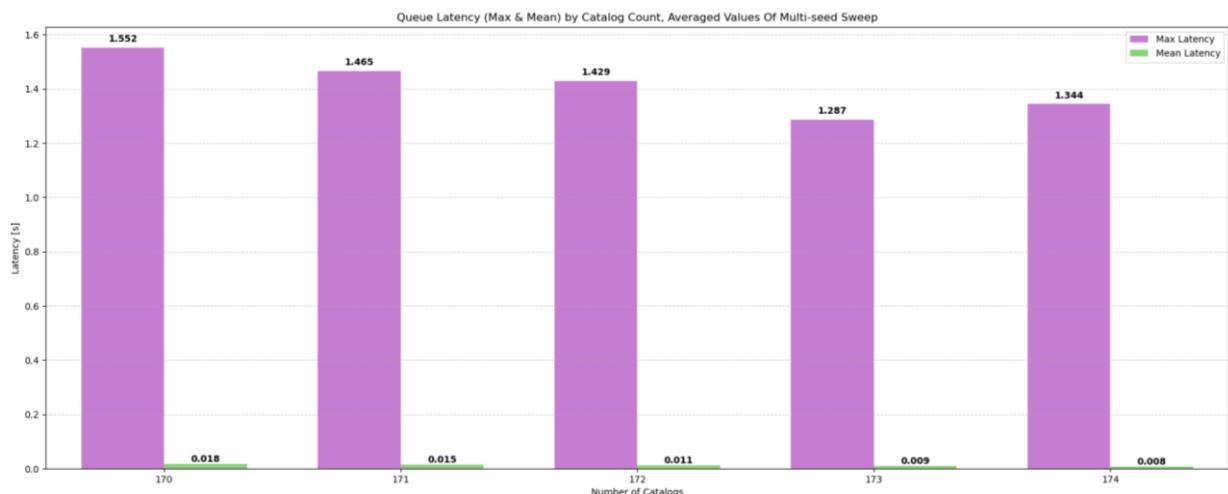
נתמקד בתחום סביב 170 קטלוגים על מנת למזער את ההשאה הכוללת של התהליך מבלי להשתמש ביותר מדדי יחידות. נרים זאת על 10 סידים שונים על מנת לבטל את ההשפעה של הסיד עליו עבדנו עד כה.

6.1.2.2 Seeds



אייר 6.13 : אורך התווך המקסימלי כפונקציה של כמות המתאים, הרצה על 10 סידים שונים.

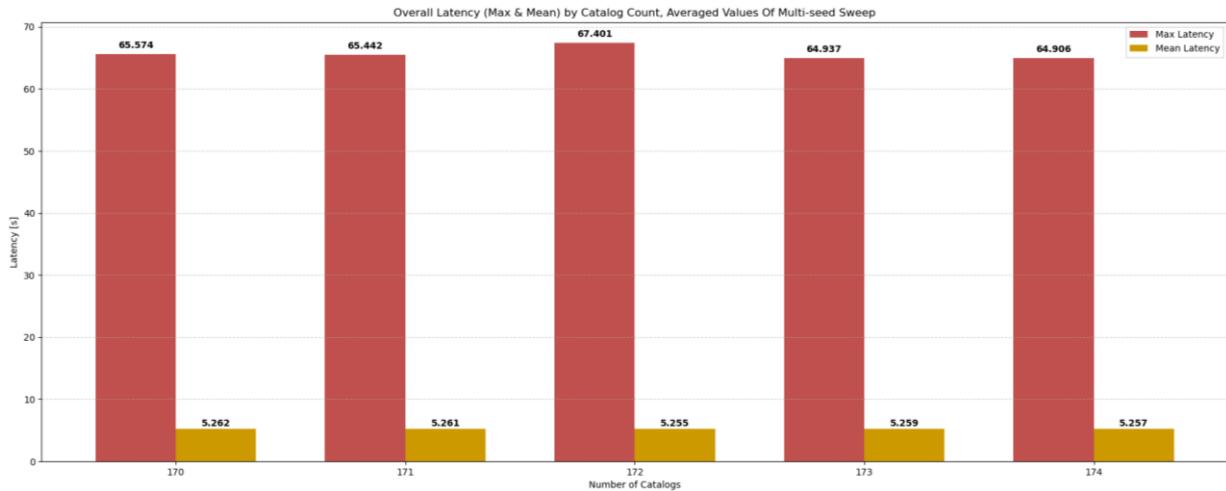
עבור אף אחת מהאפשרויות שבדקנו לא איבדנו חבילת, אך גם אורך התווך המקסימלי איינו יורד באופן עקבי עם העלאת כמות היחידות. מבין הבדיקות שעשינו, כן ניתן לראות אורך מקסימלי נמוך מ-50 בעקבות עבור 173 קטלוגים.



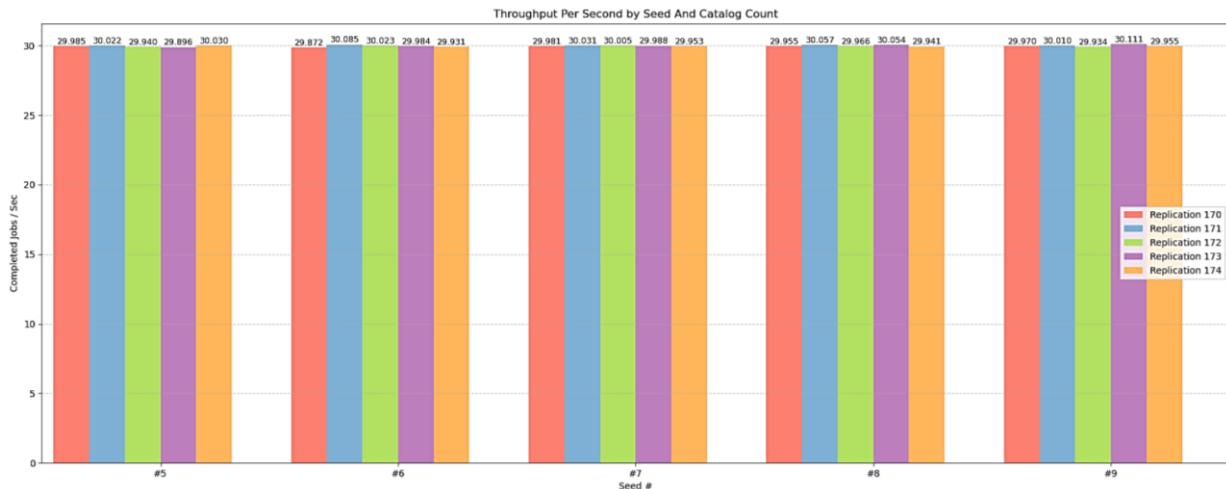
אייר 6.14 : ההשניה הממוצעת והמקסימלית בtotal עבור 170-174 קטלוגים (ממוצע ע"ג seeds שונים).

כאשר נסתכל על ההשניה המקסימלית והממוצעת (נמצע את 10 ההצלחות השונות), גם כאן עבור 173 קטלוגים מתקבלת ההשניה המקסימלית הנמוכה ביותר (דבר שתואם את אורך התווך המקסימלי הנמוך ביותר). בנוסף, גם ההשניה הממוצעת קטנה פי 2 מההשניה עבור 170 יחידות, וגדולה במליל-שנייה אחת בלבד מ-174 קטלוגים.

נסתכל על הסטטיסטיות של כלל התהיליך – מיצירת העבודות ועד לסיום הטיפול בהן:



אייר 6.15 : ההשיה הממוצעת והמקסימלית הכוללת עבור 170-174 קטלוגים (ממוצע עיג seeds שונים).



אייר 6.16 : תפוקה (Throughput) לשניה עבור 170-174 קטלוגים על גבי seeds שונים.

עבור 172 קטלוגים מתקבל ה-latency הממוצע הנמוך ביותר (5.255 msec), כאשר גם עבור שאר המדידות הממוצע נמצא בתחום של 7 ~ msec זה. לעומת זאת, ה-latency המקסימלי הנמוך ביותר מתקבל עבור 174 קטלוגים, כאשר 173 קטלוגים מתקבלת תוצאה קרובה למדיי.

נסתכל גם על התפוקה לשניה של המערכת הכוללת – נוכל לשים לב שבעור כל האפשרויות התפוקה סובבת את ה-30 עבודות לשניה, ואין כמות שמניבה תוצאות טובות בעקביות על פני האחרות.

בזה"כ, מבין המדידות, 174 קטלוגים נתונים לנו את :

- גבולות התווך הנמוכים ביותר (אורך מקסימלי, השהייה מקסימלית)
- השהייה ממוצעת (בتوוך ובתהליך כולל) בהפרש זניח מהנמוכה ביותר שנמדדה
- תפוקה לשניה זהה לשאר המדידות

בחירה בכמות קטלוגים גבוהה יותר לא משפרת את הביצועים באופן שמאזיק את העלות של שימוש בעוד, על כן נבחר להשתמש ב-174 קטלוגים.

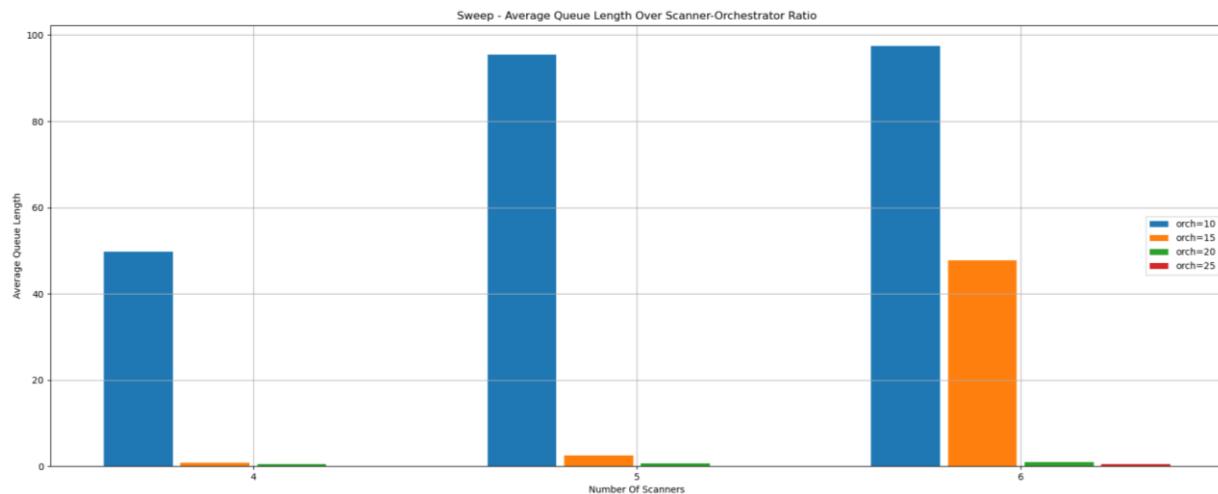
לסיכום – הצלחנו לאזן את המערכת שלנו עבור: 3 סורקים, 14 מתאימים ו-174 קטלוגים.

5 Scanners – מצב עבודה עמוס

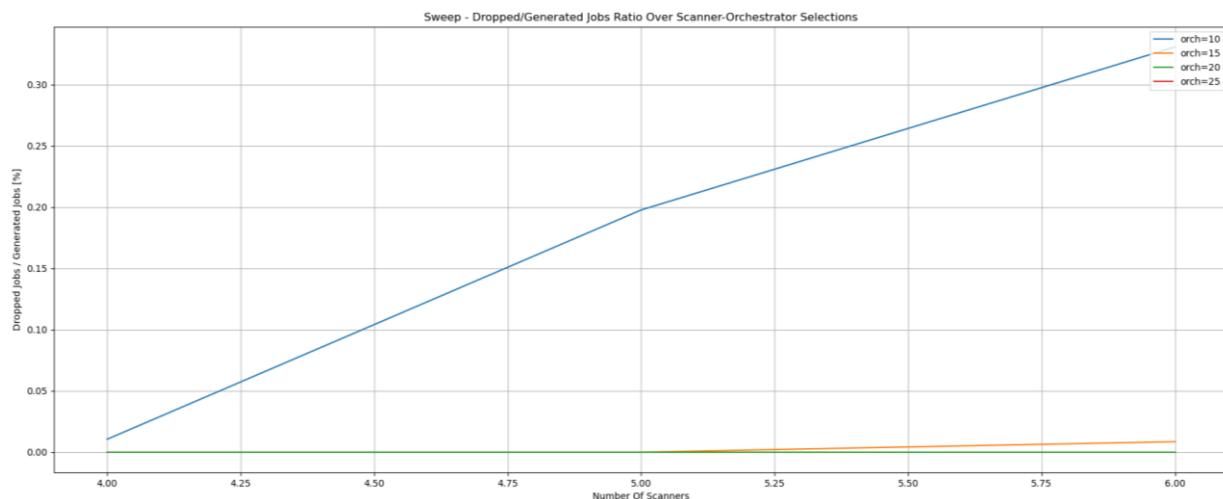
6.2.1. איזון התווך הראשון (Scanners-Orchestrators)

6.2.1.1. סריקה ראשונית כוללת

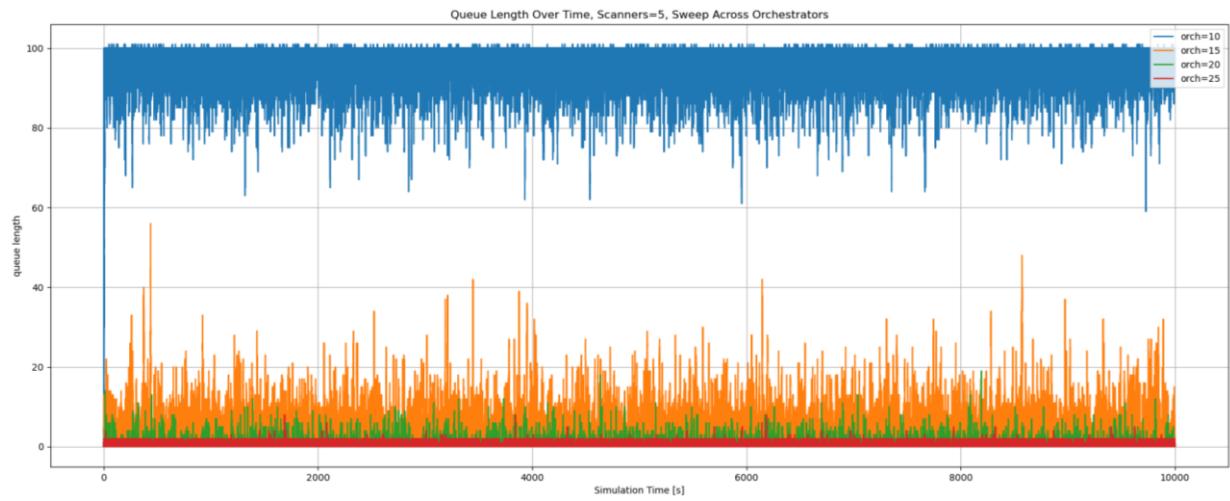
נבעץ Sweep על כמות ה-Orchestrator סיבב נקודת העבודה שלנו (נבדק בתחום 4-6), נבדוק עבור כמות היחידות הבאות : 10,15,20,25



אייר 6.17 : גרפ אורך התווך הממוצע כפונקציה של כמות ה-Scanners, לערכי Orchestrators 5,10,20,25

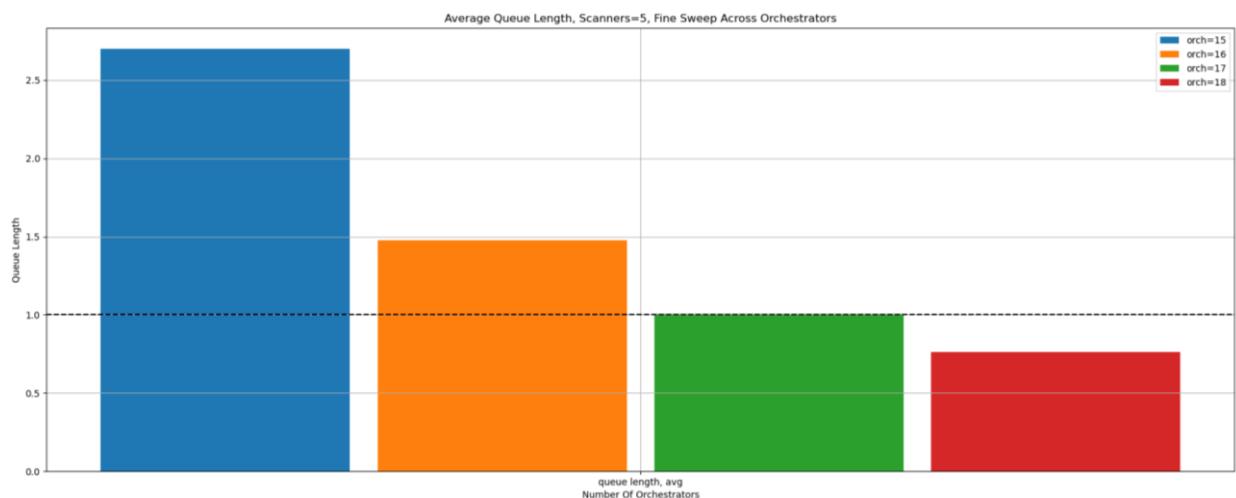


אייר 6.18 : יחס בין איבודי העבודות (dropped jobs) לכמות העבודות הכוללת כפונקציה של כמות ה-Scanners וכמות ה-Orchestrators.

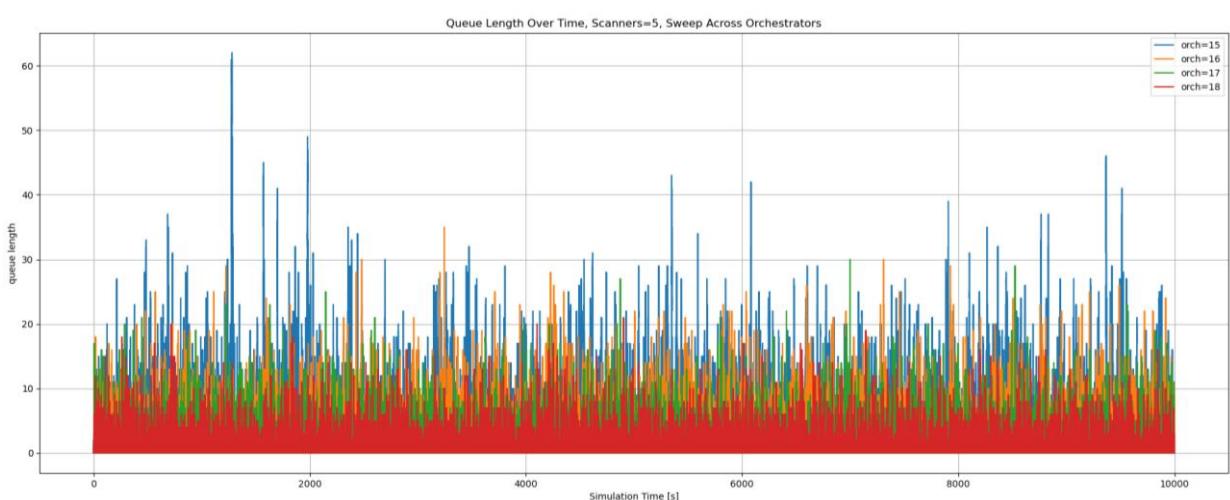


אייר 6.19 : אורך התוֹר לאותק זמן עבור 5 סורקים ו-10,15,20,25 מותאמים.

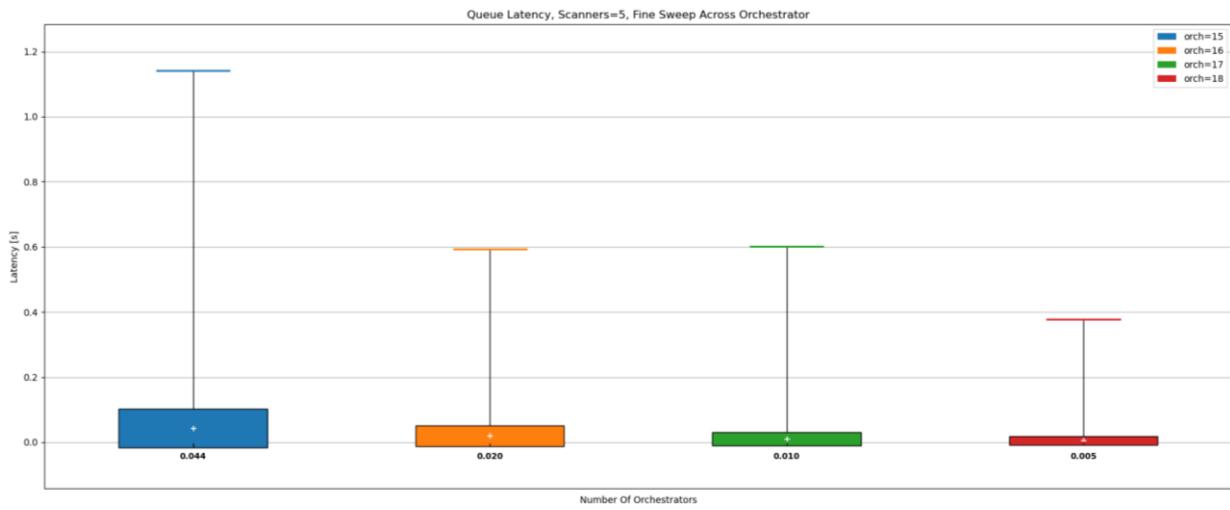
נתמך בתחום של 15-18 מותאמים (אורך תוֹר ממוצע ~, אין איבוד עבודות החל מ-15)



אייר 6.20 : אורך התוֹר הממוצע כפונקציה של כמות המותאמים. הקוו המקווקו מסמן אורך תוֹר ממוצע 1.



אייר 6.21 : אורך התוֹר לאותק זמן עבור 5 סורקים ו-15-18 מותאמים.

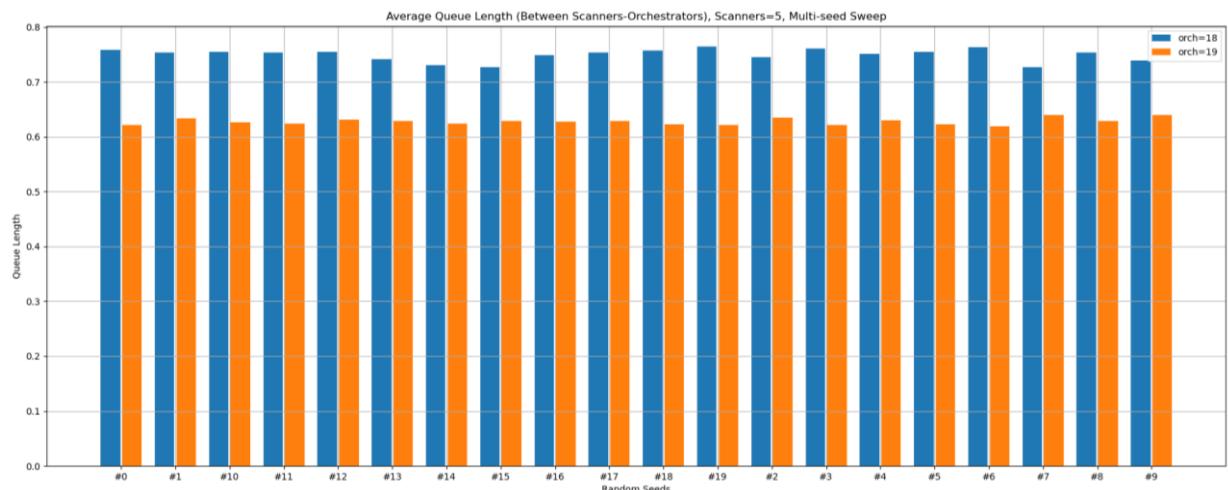


אייר 6.22 : ההשיה הממוצעת וסטיית התקן בתוור עבור 18-15 מתאים.

מתבקשת ירידה של פי 2 בהשיה הממוצעת והמקסימלית מעבר מ-17 ל-18 מתאים. נבע השוואה בין 18 ל-19 מתאים ב כדי לראות האם אנחנו ממשיכים לקבל שיפור חד כזה בתוצאות שמצדיק הוספת רכיב.

6.2.1.2. בדיקה מקיפה עם מספר seeds

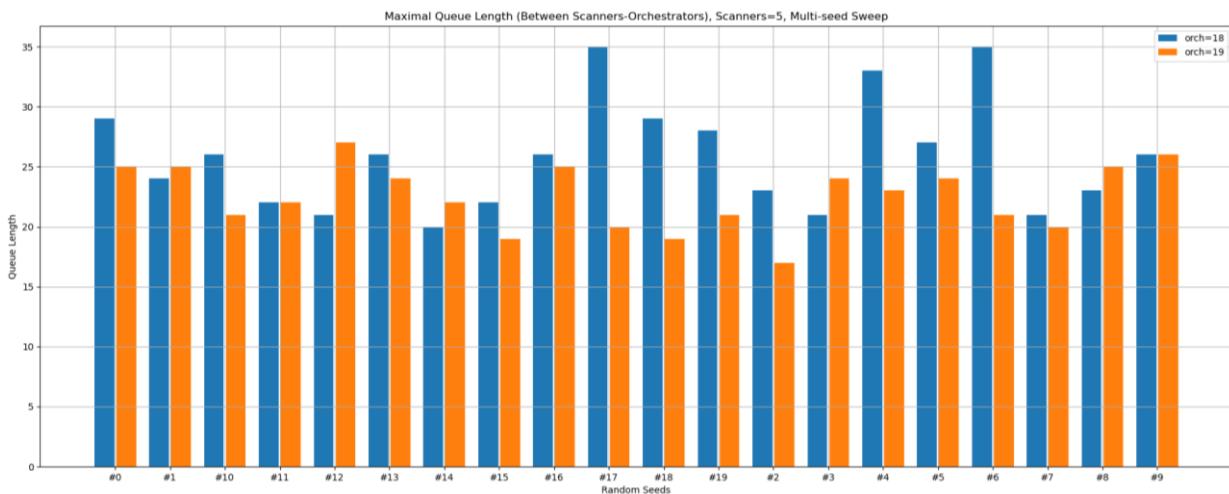
הרכנו את הסימולציה עם 20 סידים שונים עבור 17 ו-18 מתאים.



אייר 6.23 : אורך התקון הממוצע בהרצאות השונות עבור 18 ו-19 מתאים.

האורך הממוצע המתקיים עבור 18 מתאים הוא ~ 0.75 , ועבור 17 מתאים ~ 0.62 , כולל עבור שניהם נשאר היפט מתחת -1 .

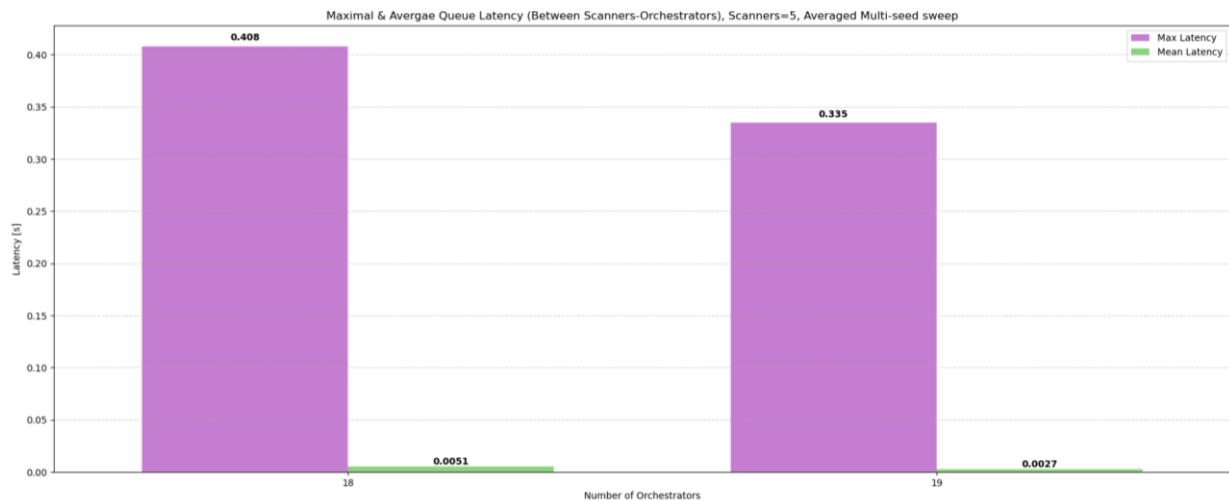
נסתכל גם על האורך המקסימלי אליו מגיעים התורדים :



אייר 6.24 : אורך התור המקסימלי בהרצות השונות עבור 18 ו-19 מתחמים.

ניתן לראות שעבור 18 מתחמים ישנו 'פיקים' גבוהים יותר של עומס (סידים 17,6,4) לעומת התחנוגות של התור עבור 19 מתחמים, שיותר 'חלקה' לאורך הסידים השונים.

נסתכל על ההשניה (המקסימלית והממוצע) ונמצע את תוצאות 20 ההרצות :



אייר 6.25 : ההשניה הממוצעת והמקסימלית בתור עבור 18 ו-19 מתחמים (ממוצע ע"ג seeds שונים).

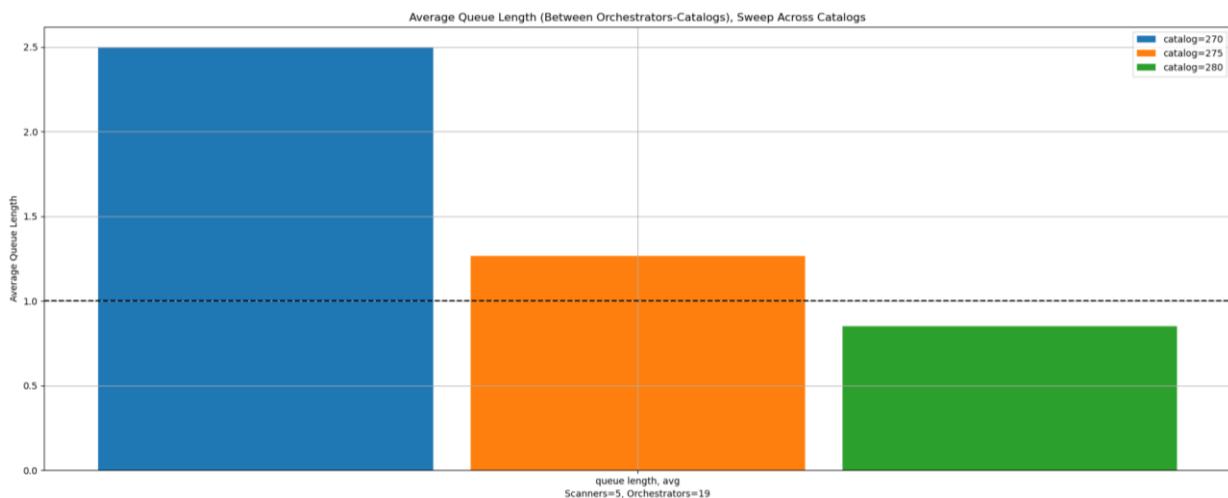
בין 18 ל-19 מתחמים יש שיפור של 22% בהשניה המקסימלית ושל 88% בהשניה הממוצעת – שיפור שווה הוספה של רכיב נוסף – לכן נבחר להשתמש במצב הסטטי ב-19 מתחמים.

6.2.2. איזון התוֹר השנִי (Orchestrators-Catalogs)

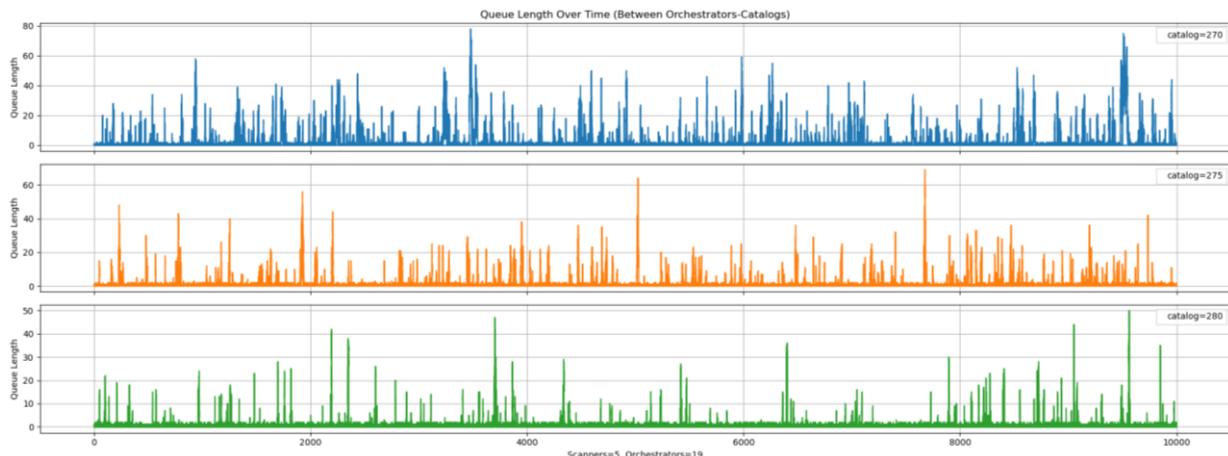
כעת נחזר את מגבלת האורך לתוֹר השנִי (100), נשתמש בכמות המתאימים שמצאו קודם (19) ונאזן את התוֹר השנִי.

6.2.2.1. סְרִיקָה רַאשׁוֹנִית

מבצע סְרִיקָה רַאשׁוֹנִית בתחום 270-280 :



אייר 6.26 : אורך התוֹר הממוצע כפונקציה של כמות הקטלוגים. הקו המכווקו מסמן אורך תוֹר ממוצע 1.



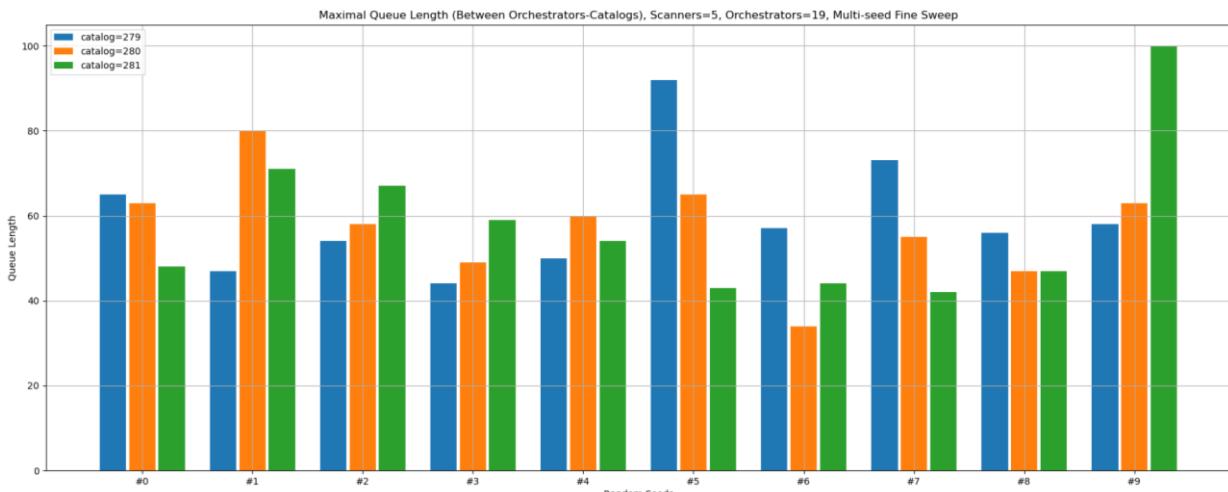
אייר 6.27 : אורך התוֹרים לאורך זמן עברו 270,275,280 קטלוגים.

מבין האפשרויות האלה, רק 280 קטלוגים עונה על הדרישה שלנו – שמירה על אורך תוֹר מקסימלי מתחת ל- 50 הודעות.

על כן, נבדוק לעומק את סביבתו (279,280,281) עם לוודא את עקביות התוצאות ונבחר את הכמות האופטימלית. זו לא בהכרח תהיה הכמות עם התוצאות הטובות ביותר, שכן כל הוספה של יחידה בכמות הינו יקרה ומשפיעה על ביצועי המערכת (סקלබיליות, העברת הודעות בין כמות גדולה של יחידות).

6.2.2.2 בדיקה מקיפה עם מספר seeds

הרכנו את הסימולציה עם 10 סידים שונים עבור 279, 280 ו- 281 קטלוגים.

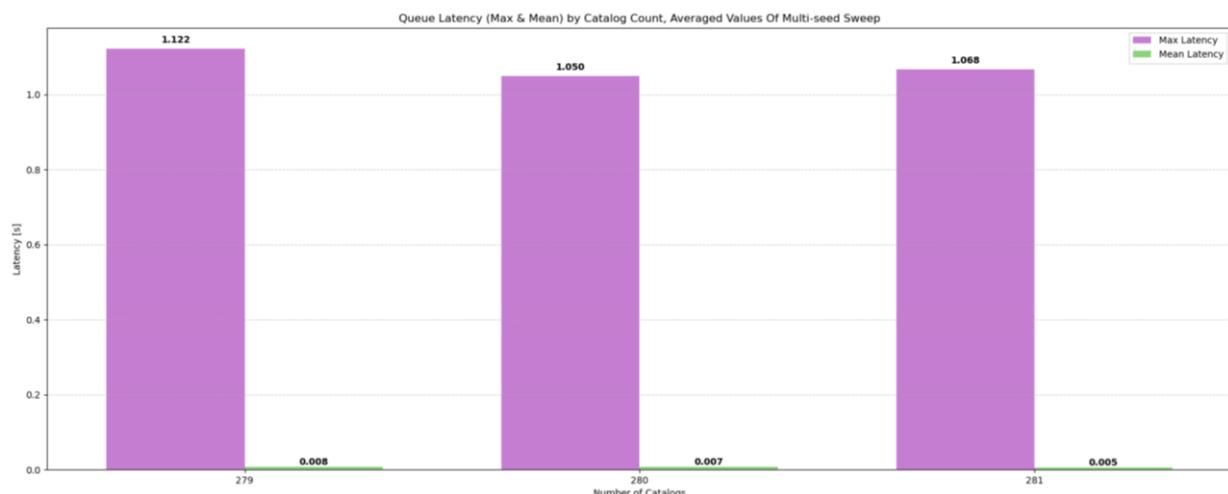


איור 6.28 : אורך התווך המקסימלי כפונקציה של כמות הקטלוגים, הרצה על 10 סידים שונים.

העלאת כמות הקטלוגים לאגרה אורךתו מקסימלי נמוך יותר לכל הסידים (כפי שניתן לראות עבור סיד 9 וסיד 5).

- עבור 281 קטלוגים התווך הגיע לשיאו לקיבולת של התווך (ללא איבוד הודעות).
- עבור 279 קטלוגים הגיעו בממוצע ל- 90% מקיבולת התווך.
- עבור 280 קטלוגים הגיעו בממוצע ל- 80% מקיבולת התווך.

בממצע, אורך התווך המקסימלי של 3 האפשרויות שבדקו הוא בסביבות 59-57 הודעות. השינוי בכמות הקטלוגים מתבטא בעיקר בהתמודדות עם 'פיקים' של עומס.



איור 6.29 : ההשיה הממוצעת והמקסימלית בתווך 279, 280, 281 קטלוגים (ממוצע עיג seeds שונים).

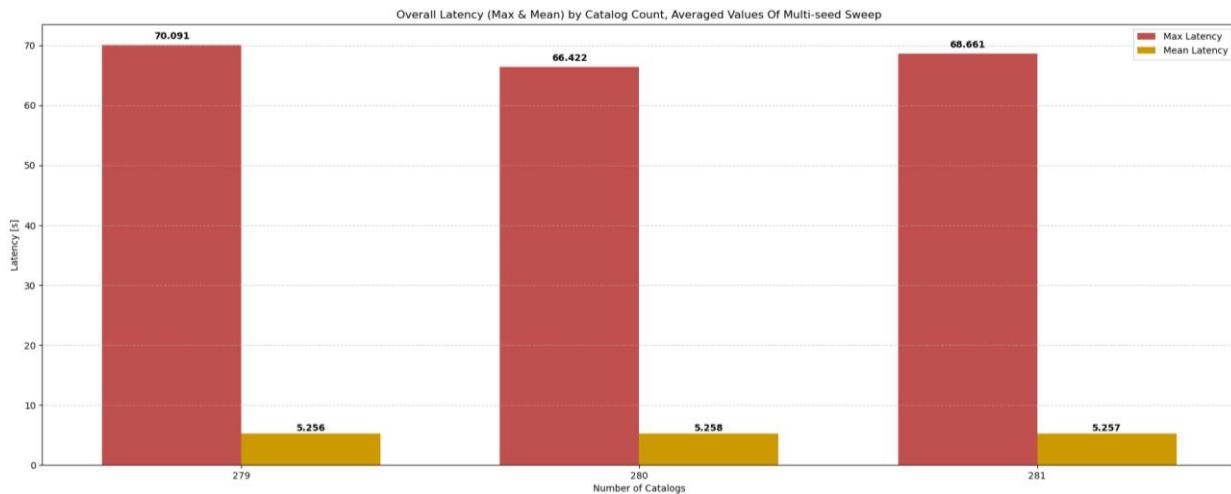
ההשיה המקסימלית הממוצעת תואמת את התוצאות עבור אורך התווך המקסימלי (ההשיה המקסימלית מתקבלת עבור מי שהגיע אחרון לתווך כשהוא הגיע לאורך הכי גבוה שלו).

מבחן ה

השחיה המומוצעת

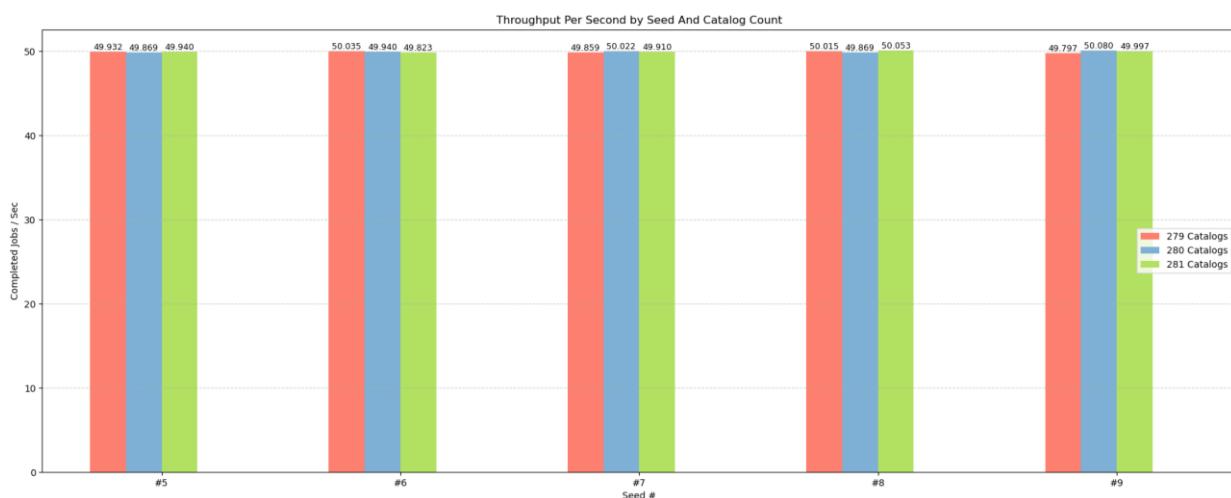
, השיפור במעבר מ-279 ל-280 קטלוגים הוא של 14%, ומעבר ל-281 משפר בעוד 40%, אם כי מדובר כבר בהשחיות ממוצעות מאוד נמוכות של מיל-שניות בודדות.

נסתכל על הסטטיסטיות של כלל התהליך – מיצירת העבודות ועד לשיום הטיפול בהן:



אייר 6.30 : השחיה המומוצעת והמקסימלית הכוללת עבור 279, 280, 281 קטלוגים (ממוצע ע"ג seeds שונים).

כאשר מסתכל על התהליך בשלהמו, **התשחיה הכוללת המומוצעת** של שלושת האופציונות שנבדקו כמעט זהה לחלווטין, בעוד שההשחיה המקסימלית נמוכה משמעותית (~3.5 שניות) עבור 280 קטלוגים לעומת 279, וכן נמוכה ביותר מ-2 שניות מ-281 קטלוגים. הוספה של עוד רכיבים לא בהכרח משפרת את הביצועים – הפערים (הקטנים) ככל הנראה נובעים מהרנדומליות של הכניסות.



אייר 6.31 : תפקה (Throughput) לשניה עבור 279, 280, 281 קטלוגים על גבי seeds שונים.

אפשר לראות שהתפקה לשנייה המתקבלת דומה מאוד בין כמות היחידות שנבדקה (כאשר משווים עבור אותו הסיד), כאשר ההפרש הכי גדול הוא של 0.1% לכל היתר (סיד 9#).

Experiment	Measurement	Replication	Module	Name	Value	Experiment	Measurement	Replication	Module	Name	Value
Orchs_Catalogs_Sweep_m	\$Catalog=279	#0	StaticNetwork.statsCollector	GeneratedJobs:count	500 799	Orchs_Catalogs_Sweep_m	\$Catalog=279	#0	StaticNetwork.statsCollector	CompletedJobs:count	500 533
Orchs_Catalogs_Sweep_m	\$Catalog=279	#1	StaticNetwork.statsCollector	GeneratedJobs:count	500 151	Orchs_Catalogs_Sweep_m	\$Catalog=279	#1	StaticNetwork.statsCollector	CompletedJobs:count	499 894
Orchs_Catalogs_Sweep_m	\$Catalog=279	#2	StaticNetwork.statsCollector	GeneratedJobs:count	499 605	Orchs_Catalogs_Sweep_m	\$Catalog=279	#2	StaticNetwork.statsCollector	CompletedJobs:count	499 326
Orchs_Catalogs_Sweep_m	\$Catalog=279	#3	StaticNetwork.statsCollector	GeneratedJobs:count	499 764	Orchs_Catalogs_Sweep_m	\$Catalog=279	#3	StaticNetwork.statsCollector	CompletedJobs:count	499 511
Orchs_Catalogs_Sweep_m	\$Catalog=279	#4	StaticNetwork.statsCollector	GeneratedJobs:count	501 281	Orchs_Catalogs_Sweep_m	\$Catalog=279	#4	StaticNetwork.statsCollector	CompletedJobs:count	501 003
Orchs_Catalogs_Sweep_m	\$Catalog=279	#5	StaticNetwork.statsCollector	GeneratedJobs:count	499 565	Orchs_Catalogs_Sweep_m	\$Catalog=279	#5	StaticNetwork.statsCollector	CompletedJobs:count	499 316
Orchs_Catalogs_Sweep_m	\$Catalog=279	#6	StaticNetwork.statsCollector	GeneratedJobs:count	500 648	Orchs_Catalogs_Sweep_m	\$Catalog=279	#6	StaticNetwork.statsCollector	CompletedJobs:count	500 353
Orchs_Catalogs_Sweep_m	\$Catalog=279	#7	StaticNetwork.statsCollector	GeneratedJobs:count	498 875	Orchs_Catalogs_Sweep_m	\$Catalog=279	#7	StaticNetwork.statsCollector	CompletedJobs:count	498 594
Orchs_Catalogs_Sweep_m	\$Catalog=279	#8	StaticNetwork.statsCollector	GeneratedJobs:count	500 422	Orchs_Catalogs_Sweep_m	\$Catalog=279	#8	StaticNetwork.statsCollector	CompletedJobs:count	500 147
Orchs_Catalogs_Sweep_m	\$Catalog=279	#9	StaticNetwork.statsCollector	GeneratedJobs:count	498 261	Orchs_Catalogs_Sweep_m	\$Catalog=279	#9	StaticNetwork.statsCollector	CompletedJobs:count	497 973
Orchs_Catalogs_Sweep_m	\$Catalog=280	#0	StaticNetwork.statsCollector	GeneratedJobs:count	499 321	Orchs_Catalogs_Sweep_m	\$Catalog=280	#0	StaticNetwork.statsCollector	CompletedJobs:count	499 044
Orchs_Catalogs_Sweep_m	\$Catalog=280	#1	StaticNetwork.statsCollector	GeneratedJobs:count	500 151	Orchs_Catalogs_Sweep_m	\$Catalog=280	#1	StaticNetwork.statsCollector	CompletedJobs:count	499 894
Orchs_Catalogs_Sweep_m	\$Catalog=280	#2	StaticNetwork.statsCollector	GeneratedJobs:count	500 513	Orchs_Catalogs_Sweep_m	\$Catalog=280	#2	StaticNetwork.statsCollector	CompletedJobs:count	500 250
Orchs_Catalogs_Sweep_m	\$Catalog=280	#3	StaticNetwork.statsCollector	GeneratedJobs:count	499 648	Orchs_Catalogs_Sweep_m	\$Catalog=280	#3	StaticNetwork.statsCollector	CompletedJobs:count	499 399
Orchs_Catalogs_Sweep_m	\$Catalog=280	#4	StaticNetwork.statsCollector	GeneratedJobs:count	500 347	Orchs_Catalogs_Sweep_m	\$Catalog=280	#4	StaticNetwork.statsCollector	CompletedJobs:count	500 096
Orchs_Catalogs_Sweep_m	\$Catalog=280	#5	StaticNetwork.statsCollector	GeneratedJobs:count	498 935	Orchs_Catalogs_Sweep_m	\$Catalog=280	#5	StaticNetwork.statsCollector	CompletedJobs:count	498 686
Orchs_Catalogs_Sweep_m	\$Catalog=280	#6	StaticNetwork.statsCollector	GeneratedJobs:count	499 634	Orchs_Catalogs_Sweep_m	\$Catalog=280	#6	StaticNetwork.statsCollector	CompletedJobs:count	499 396
Orchs_Catalogs_Sweep_m	\$Catalog=280	#7	StaticNetwork.statsCollector	GeneratedJobs:count	500 471	Orchs_Catalogs_Sweep_m	\$Catalog=280	#7	StaticNetwork.statsCollector	CompletedJobs:count	500 215
Orchs_Catalogs_Sweep_m	\$Catalog=280	#8	StaticNetwork.statsCollector	GeneratedJobs:count	498 934	Orchs_Catalogs_Sweep_m	\$Catalog=280	#8	StaticNetwork.statsCollector	CompletedJobs:count	498 691
Orchs_Catalogs_Sweep_m	\$Catalog=280	#9	StaticNetwork.statsCollector	GeneratedJobs:count	501 067	Orchs_Catalogs_Sweep_m	\$Catalog=280	#9	StaticNetwork.statsCollector	CompletedJobs:count	500 802
Orchs_Catalogs_Sweep_m	\$Catalog=281	#0	StaticNetwork.statsCollector	GeneratedJobs:count	500 570	Orchs_Catalogs_Sweep_m	\$Catalog=281	#0	StaticNetwork.statsCollector	CompletedJobs:count	500 309
Orchs_Catalogs_Sweep_m	\$Catalog=281	#1	StaticNetwork.statsCollector	GeneratedJobs:count	500 098	Orchs_Catalogs_Sweep_m	\$Catalog=281	#1	StaticNetwork.statsCollector	CompletedJobs:count	499 840
Orchs_Catalogs_Sweep_m	\$Catalog=281	#2	StaticNetwork.statsCollector	GeneratedJobs:count	499 556	Orchs_Catalogs_Sweep_m	\$Catalog=281	#2	StaticNetwork.statsCollector	CompletedJobs:count	499 251
Orchs_Catalogs_Sweep_m	\$Catalog=281	#3	StaticNetwork.statsCollector	GeneratedJobs:count	500 132	Orchs_Catalogs_Sweep_m	\$Catalog=281	#3	StaticNetwork.statsCollector	CompletedJobs:count	499 857
Orchs_Catalogs_Sweep_m	\$Catalog=281	#4	StaticNetwork.statsCollector	GeneratedJobs:count	498 825	Orchs_Catalogs_Sweep_m	\$Catalog=281	#4	StaticNetwork.statsCollector	CompletedJobs:count	498 545
Orchs_Catalogs_Sweep_m	\$Catalog=281	#5	StaticNetwork.statsCollector	GeneratedJobs:count	499 647	Orchs_Catalogs_Sweep_m	\$Catalog=281	#5	StaticNetwork.statsCollector	CompletedJobs:count	499 400
Orchs_Catalogs_Sweep_m	\$Catalog=281	#6	StaticNetwork.statsCollector	GeneratedJobs:count	498 503	Orchs_Catalogs_Sweep_m	\$Catalog=281	#6	StaticNetwork.statsCollector	CompletedJobs:count	498 230
Orchs_Catalogs_Sweep_m	\$Catalog=281	#7	StaticNetwork.statsCollector	GeneratedJobs:count	499 376	Orchs_Catalogs_Sweep_m	\$Catalog=281	#7	StaticNetwork.statsCollector	CompletedJobs:count	499 097
Orchs_Catalogs_Sweep_m	\$Catalog=281	#8	StaticNetwork.statsCollector	GeneratedJobs:count	500 782	Orchs_Catalogs_Sweep_m	\$Catalog=281	#8	StaticNetwork.statsCollector	CompletedJobs:count	500 332
Orchs_Catalogs_Sweep_m	\$Catalog=281	#9	StaticNetwork.statsCollector	GeneratedJobs:count	500 358	Orchs_Catalogs_Sweep_m	\$Catalog=281	#9	StaticNetwork.statsCollector	CompletedJobs:count	499 966

טבלה 6.4 : כמות העבודות שיוצרו בסה"כ בכל סימולציה (שמאל), כמות העבודות שהושלמו בסה"כ בכל סימולציה (ימין). (שאר העבודות שיוצרים וטרם הושלמו נמצאות בעבודה ברגע עצירת הסימולציה)

בסה"כ, בין האפשרויות שנבדקו:

- אין הבדל משמעותי בתפוקה לשנייה, בהשניה המומוצעת ובאורך התור המומוצע.
- ההבדלים באים לידי ביטוי בהתמודדות עם מצב קצה – אורך התור המקסימלי וההשניה המקסימלית, עם יתרון מובהק לבחירה של 280 קטלוגים.

לכן, כדי להימנע ממצב של התמלאות של התור (שימוש נייד לкриישה של כלל המערכת האמיתית), וכידי לקבל את הביצועים הטובים ביותר מבחן השניה, נבחר להשתמש ב- 280 קטלוגים.

לסיכום – הצלחנו לאוזן את המערכת שלנו עבור: 5 סורקים, 19 מתאימים ו- 280 קטלוגים.

7. תוצאות - סימולציה דינמית

נתמקד ב-2 פונקציות שונות לבקרה של המערכת.

1. עליה לינארית, ירידה עפ"י היסטוריזס (מייצוע X דגימות אחרות).
2. מנגנון שיעורץ זמן לריקון התור (Queue Drain-rate Estimation). (Queue Drain-rate Estimation)

את התוצאות נבחן על בסיס אותם המדדים כמו קודם:

- i. כמות הרכיבים הפעילים
- ii. השהיה (Latency)
- iii. תפקה (Throughput) – כמות עבודות לדקה
- iv. תנודתיות המנגנון (כמו 'פתיחות' ו'סגורות' של רכיבים מתבצעות לאורך הסימולציה)
- v. איבודי הודעות (כ-% מכלל ההודעות)
- vi. אורך תורים ממוצע

את כל התוצאות נרים עיג 5 סידים שונים ונמצע את התוצאות. את כל המדדים נרמל בין 0 ל-1, כאשר 1 זו התוצאה הכי טובה מבין התוצאות שהתקבלו ו-0 הכי גרוע.

נבחר פונקציית ציון שמספקת את כל המדדים האלו (מנורמלים בין 0 ל-1):

$$\begin{aligned}Score = & 0.2 \cdot (\text{max latency}) + 1 \cdot (\text{avg. latency}) + 0.5 \cdot (\text{avg. orch num}) + 0.5 \cdot (\text{avg. catalog num}) \\& + 0.25 \cdot (\text{throughput}) + 0.2 \cdot (\text{avg. queues length}) + 0.3 \cdot (\text{drop rate}) + 0.25 \\& \cdot (\text{control messages})\end{aligned}$$

אנחנו שמים משקל גבוה יותר להשהיות ולכמות הרכיבים הדלקים במערכת, שכן המטרה המרכזית של הבקרה הדינמית היא הקטנת **כמות הרכיבים במערכת מבליל** לגרום לפגיעה גדולה בBITS.

הערה: הציון המרבי שניתן להציג מפונקציה זו הוא 3.2.

לשתי פונקציות הבקרה, נבחר את קומבינציה הפרמטרים האופטימלית (לפי חישוב הציון הנ"ל), ולבסוף נבצע השוואה בין שתי התוצאות שהתקבלו.

1.7. סקר ביצועים – עלייה לינארית, רידעה עפ"י היסטורייז

נזכיר את מנגנון הבדיקה המוטמע בסימולציה:

$$\text{Number of pods} = (\text{Queue length}) * \frac{(\text{Maximal pods}) - (\text{Minimal pods})}{\text{Queue Threshold}} + (\text{Minimal pods})$$

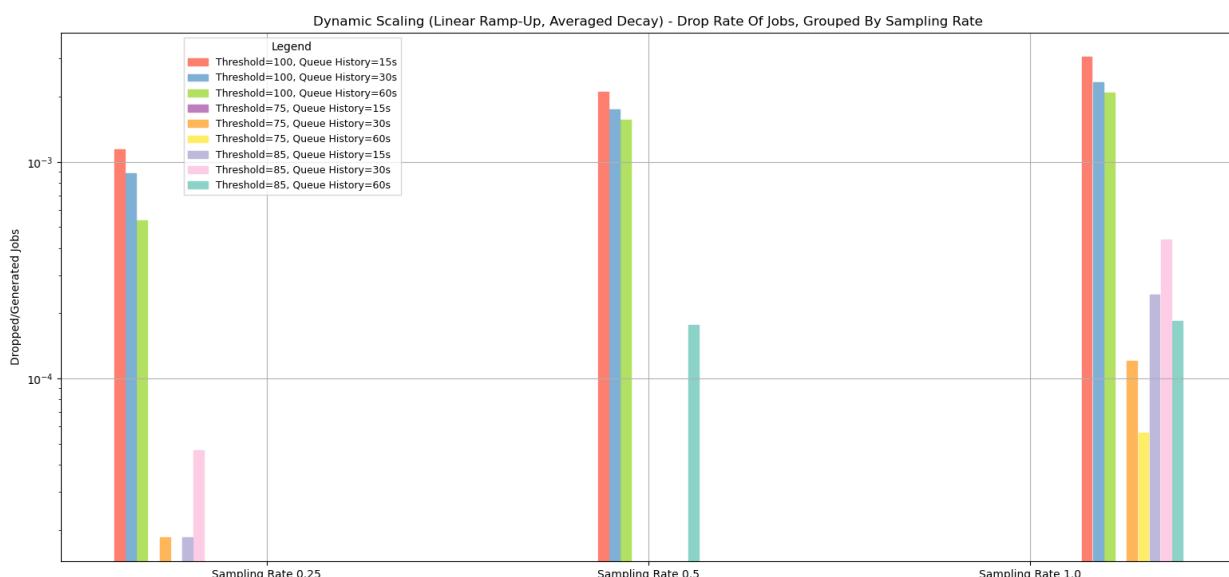
כאשר Queue Threshold – אורך התור שמננו והלאה אנחנו מפעילים את מקסימום היחידות.

- אם תוצאת החישוב גדולה מכמות הרכיבים הפעילים – מעלים את כמות הרכיבים לערך שהתקבל עד לערך המקסימלי).
- אם הכמות קטנה – ממצאים את X הדגימות האחרונות ומורידים לכמות המתאימה עפ"י הנוסחה.

בצע סקר ביצועים על הפרמטרים הבאים:

- כמות פודים מינימלית : 1 (עבור מתאימים וקטלוגים).
- כמות פודים מקסימלית : עבור מתאימים – 14, עבור קטלוגים – 174. (מחסימולציה הסטטistica).
- אורך התור עבורו מפעילים את כל הפודים : 00,75,85,100.
- אורך ההיסטוריה אותה ממצאים (Sampling History) : 15,30,60 : 15 שניות.
- קצב הדגימה (כל כמה זמן דוגמים את אורכי התורים) : 1, 0.5, 0.25 שניות.

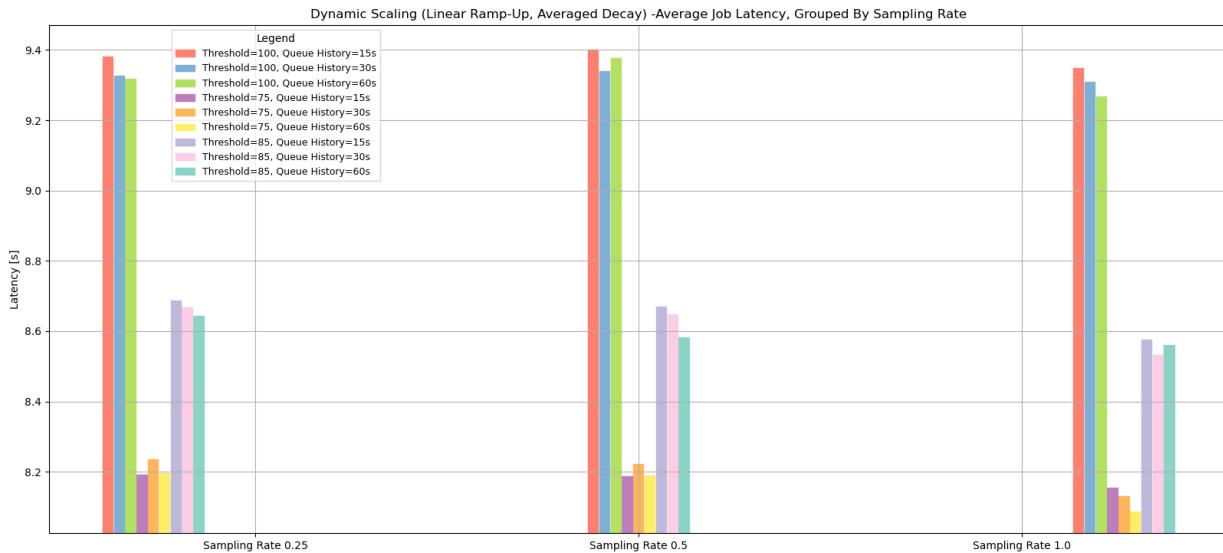
7.1.1. תוצאות עבור Sweep יחיד – תוצאות לא מנורמלות



אייר 1.7: שיעור איבוד ההודעות עבור כל הקומבינציות של הפרמטרים הנבדקים, סימ' דינמית (עלייה לינארית, רידעה עפ"י היסטורייז).

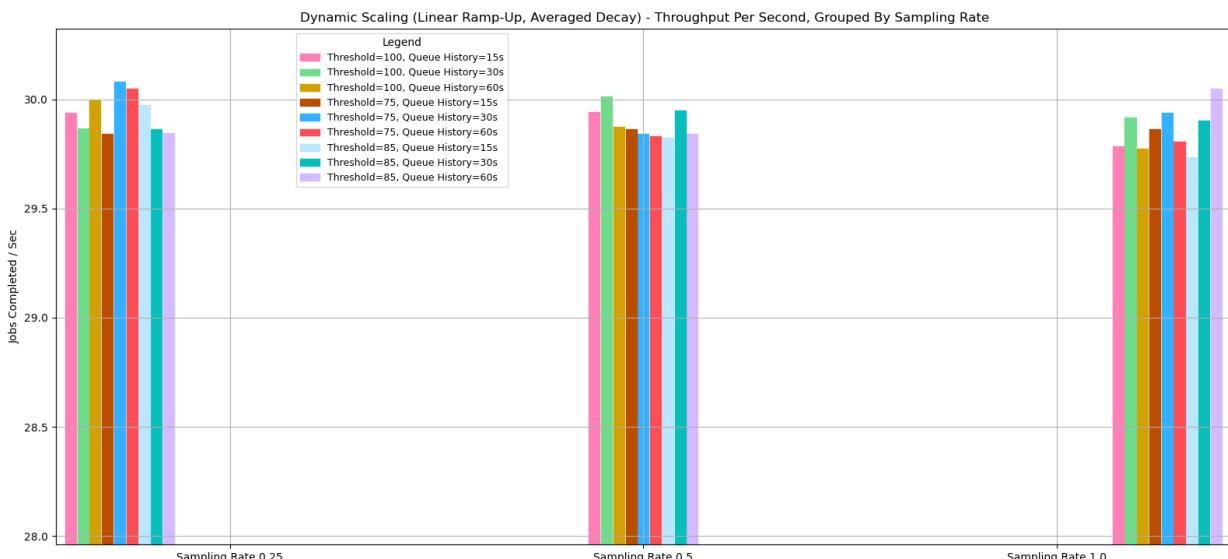
ניתן לראות את ההשפעה של כל הפרמטרים שנבדקים על איבוד ההודעות. כאשר ההשפעה הגדולה ביותר היא דואקע עי' threshold – שמאפשר להגיע למקסימום רכיבים בשלב מוקדם יותר של העומס בתור, ובכך מקטין משמעותית את איבוד החרבויות. הפרמטר השני ביכולתו להשפיע הינו קצב הדגימה, שמתקשר באופן ישיר גם לthreshold. לבסוף, גם לאורך ההיסטוריה ישנה השפעה, אם כי מועטה יותר, ביחס הפוך

לאיבוד ההודעות. דבר זה נובע מכך שאנחנו פחות תנודתיים עם אורך ההיסטוריה גדול יותר, ובכך פחות נוטים להגיע למצב של איבוד.



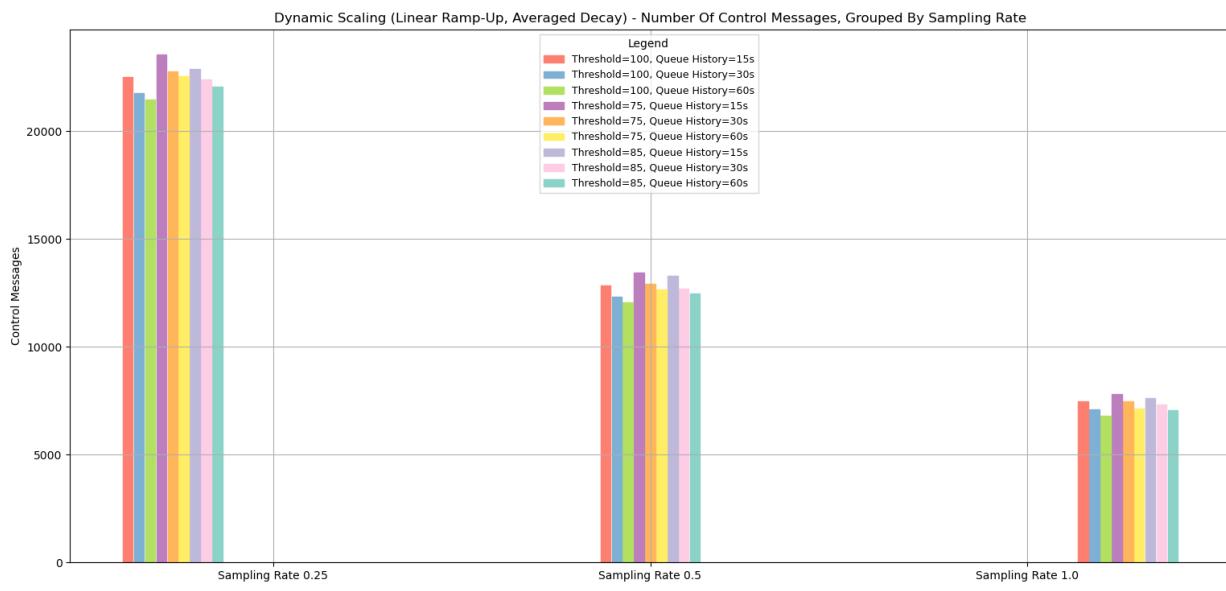
אייר 7.2 : השהיה ממוצעת במערכת עבור כל הקומבינציות של הפרמטרים הנבדקים, סימן דינמית (עליה לינארית, רידזה עפ"י היסטורי).

בניגוד לשיעור איבוד ההודעות, ההשיה כמעט ולא מושפעת מאריך ההיסטוריה – אלא מושפעת בצורה ניכרת מבחירה ה-threshold – ככל שהוא נמוך יותר, כך יש יותר רכיבים פעילים וההשיה קטנה בהתאם.



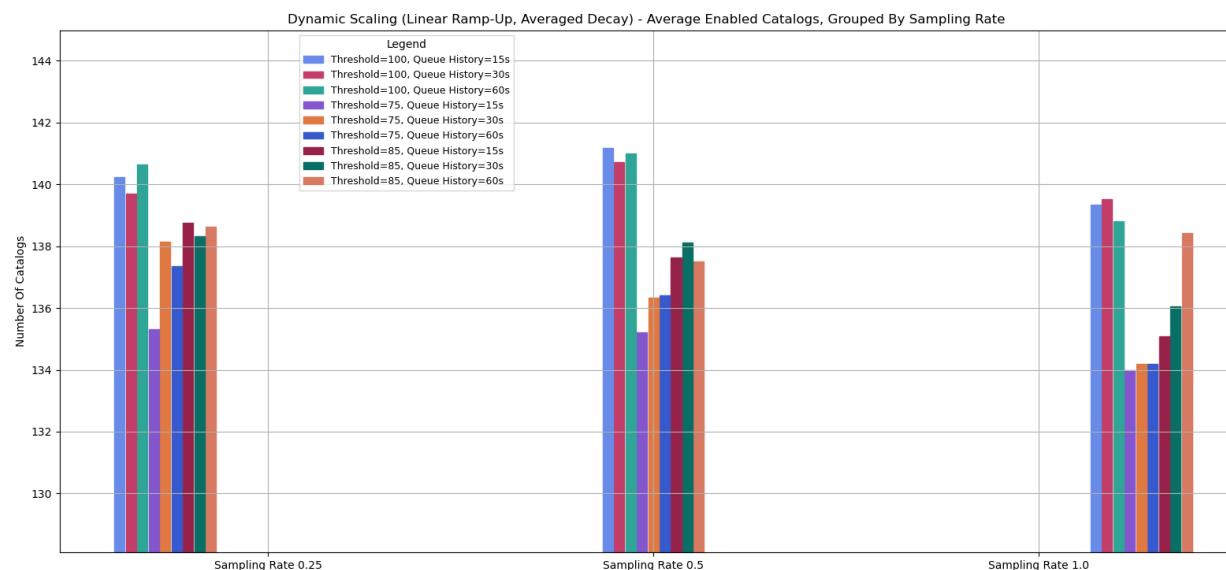
אייר 7.3 : תפוקה (עבודות לדקה) עבור כל הקומבינציות של הפרמטרים הנבדקים, סימן דינמית (עליה לינארית, רידזה עפ"י היסטורי).

התפוקה מושפעת בעיקר מקצב הדגימה – ככל שקצב הדגימה גבוה יותר, כך התפוקה גבוהה יותר. זאת מכיוון שעבור קצב דגימה גבוהה יותר, המערכת יכולה מגיבה מהר יותר לשינויים בעומס ומתאימה את כמות היחידות לעומס, לעומת קצב דגימה איטי יותר. ההבדלים הינם קטנים באופן יחסית (הפרש של 0.5 עבודה לשנייה), אך מctrברים לאורך זמן.



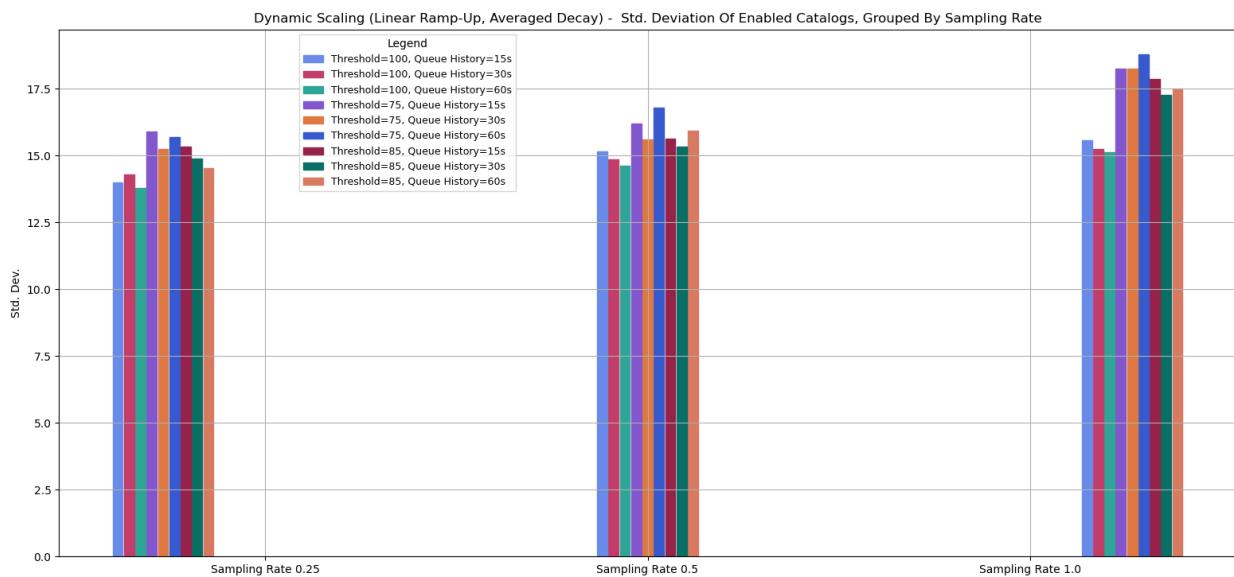
אייר 4.7 : כמות הודעות הבקרה (הפעלה/כיבוי ייחדות) עבור כל הקומבינציות של הפרמטרים הנבדקים, סימן דינמית (עליה לינארית, ירידה עפ"י היסטורי).

מדד זה מהוות בעיקר Sanity Check למימוש שלנו לפונקציית הבקרה - ככל שקצב הדגימה גבוה יותר, כך כמות הודעות הבקרה שנשלחות גדולה. ניתן לראות קשר כמעט לינארי ביןיהם, ככלمر כמעט בכל 'דגם' המערכת מגיבה ושולחת הודעות בקרה. בנוסף, הגדלת ההיסטוריה מפחיתה את גם היא את כמות הודעות הבקרה – לוקח יותר זמן עד שמתחלת הפחתה של ייחדות, אם בכלל היא מתאפשרת עם מיצוע ארוך.



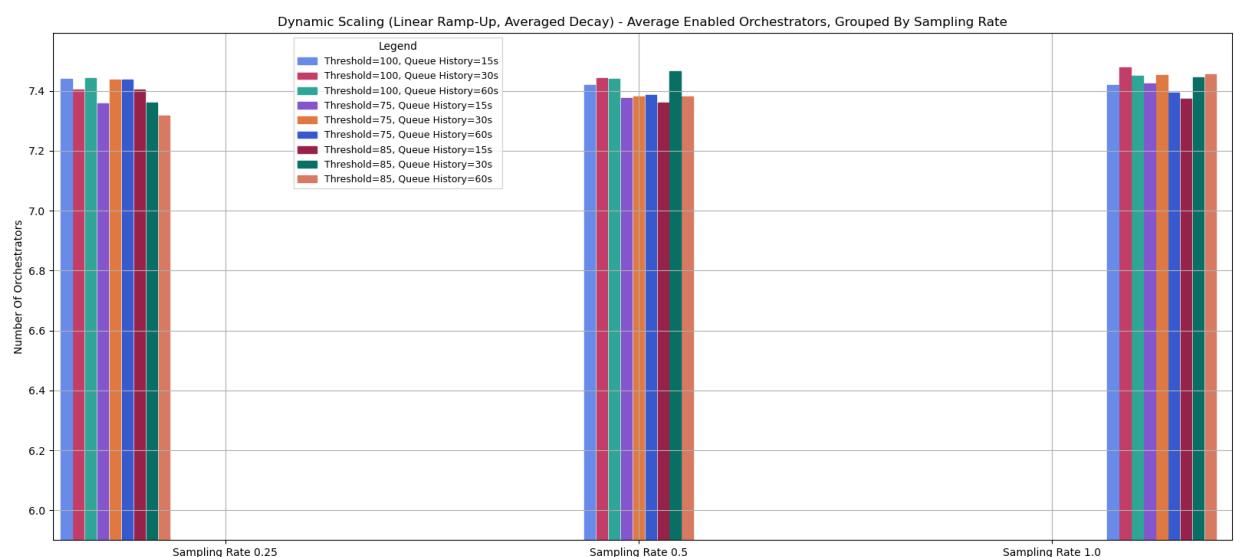
אייר 5.7 : כמות הקטלוגים המופעלים ממוצע עבור כל הקומבינציות של הפרמטרים הנבדקים, סימן דינמית (עליה לינארית, ירידה עפ"י היסטורי).

ניתן לראות כאן תופעה מעניינת: בקצב הדגימה האיטי ביותר (כל שנייה אחת), עבור הערך הנוכחי (75) והכי גבוה (100) מתקבלת כמות הקטלוגים ממוצעת דומה, אך עבור threshold של 85 יש הבדלים גדולים בין התוצאות השונות בהתאם לאורך ההיסטוריה. נראה שמדובר בנקודת גבול עדינה שבה אורך ההיסטוריה משפייע משמעותית. אפשר לשער שהזנה נובע מ-“bursts” סמוכים בזמן שנכנסו או לא נכנסו לחולון המיצוע בסימולציות השונות.



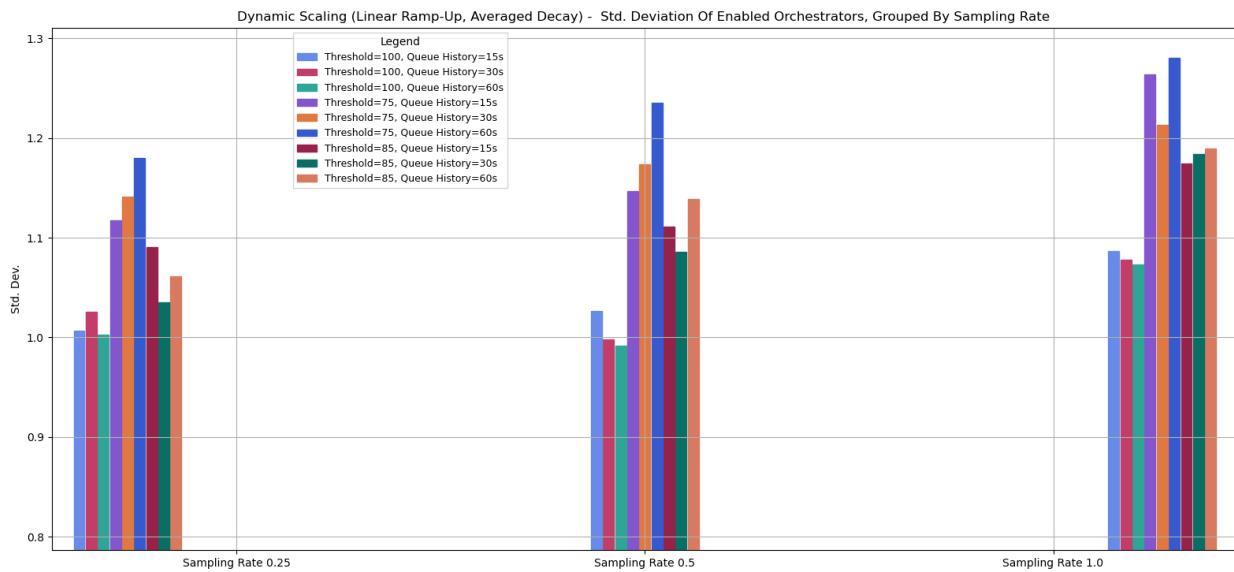
אייר 7.6 : סטיית תקן עבור כמהות הקטלוגים המופעלים לכל הקומבינציות של הפרמטרים הנבדקים, סימני דינמית (עליה לינארית, ירידה עפ"י היסטורי).

ניתן לראות שככל שקצב הדגימה איטי יותר, סטיית התקן עולה, ככל יותר הרכיבים פחות יציבה. עבורו ניתן לראות שבדיל של 75 thresholds, אך ב-sd-values קטנים, אך ב-sd-values גבוהים יותר.



אייר 7.7 : כמהות המתאיםים המופעלים בממוצע עבור כל הקומבינציות של הפרמטרים הנבדקים, סימני דינמית (עליה לינארית, ירידה עפ"י היסטורי).

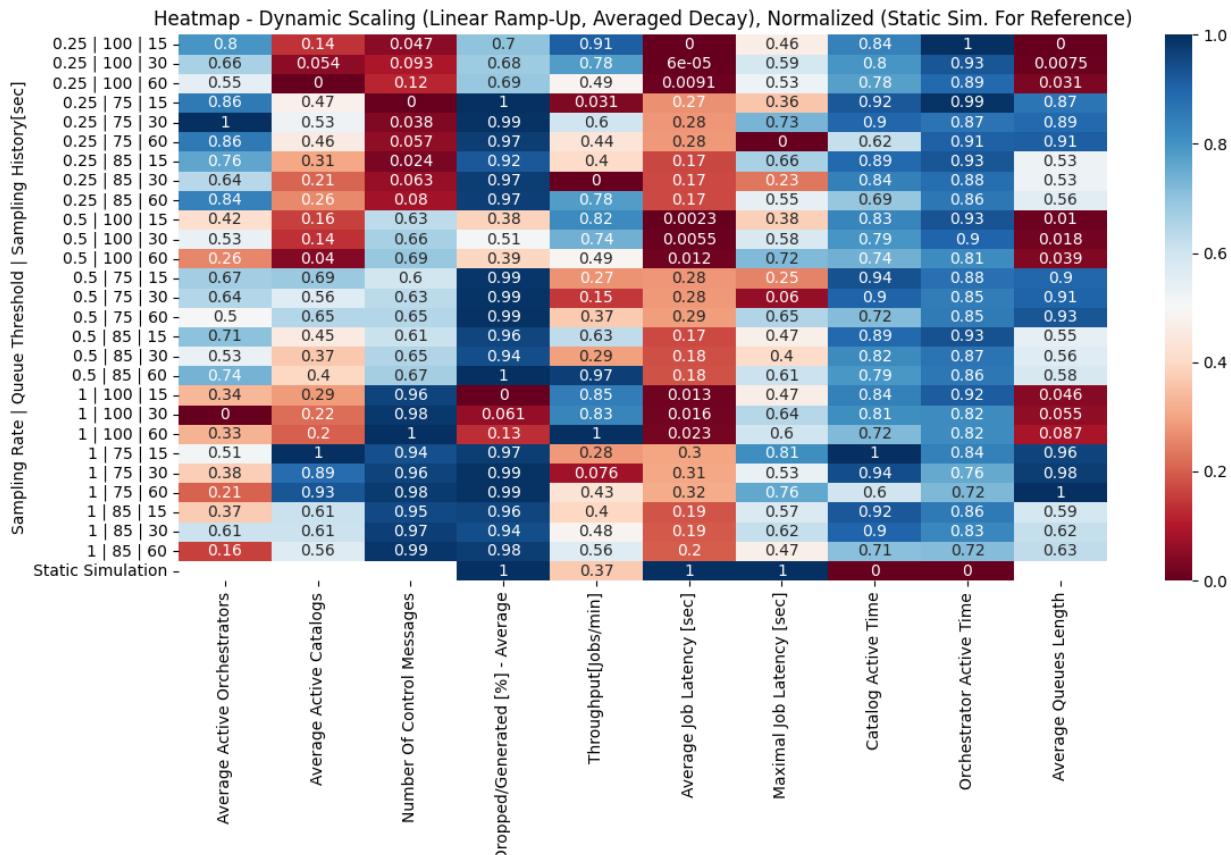
בשונה מבאיור 7.5, כמהות המתאיםים הממוצעת מגיעה לשינוי של הפרמטרים, ודומה למדיי בין כל הקומבינציות שנבדקו. נראה שהתור הזה מתייצב על ערך זזה עבור כל הקומבינציות, עם הטיה קלה (של יותר רכיבים) כאשר קצב הדגימה הcy איטי (כל שנייה).



אייר 7.8: סטיית תקן עבור כמות המתאמים המופעלים לכל הקומבינציות של הפרמטרים הנבדקים, סימני דינמיות (עליה לינארית, ירידה עפ"י היסטורי).

ישנה תנודתיות גבוהה יותר במספר המתאים כאשר קצב הדגימה הינו האיטי ביותר (אחת לשניה). בנוסף, בכל קצב הדגימה, סטיית התקן עבור אורך היסטוריה של 60 שניות עם threshold של 75 גבוהה במיוחד – דבר זה מעיד על **תנודתיות חזה יותר בכמות הרכיבים** בקומבינציה זו – ככלומר, המערכת נוטה להגיע לערכים קיצוניים יותר, כנראה משום שלוקח לה זמן רב יותר להתאושש ולהפחית את כמות הרכיבים לאחר שהיא הייתה בעומס. למרות זאת, הערך הממוצע שמתקיים בכל קבוצה (עפ"י קצב דגימה) כמעט זהה – משמע שהמערכת באופן כלל מגיעה לאוות איזון, פשוט בדרכים שונות.

7.1.2. תוצאות כוללות – Multiple Sweeps

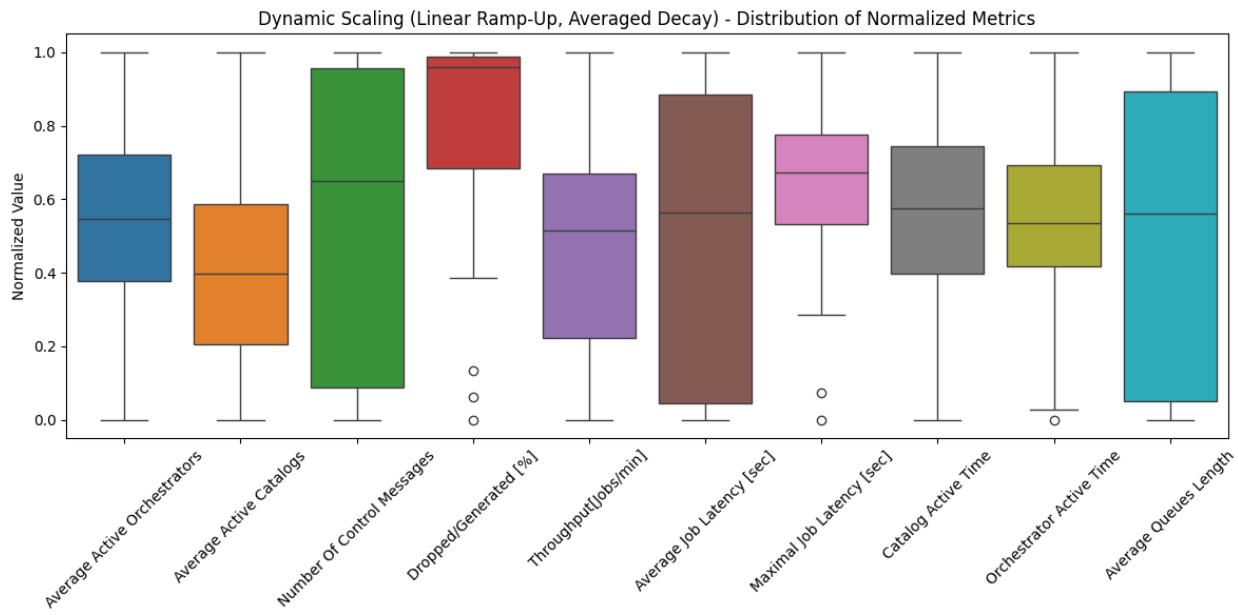


אייר 7.9 מנורמל לסדרת הפרמטרים עבור הסימני הדינמי (עליה לינארית, ירידה עפ"י היסטריז), ביחס לסימולציה הステטיסטית (שורה תחתונה).

כמה תובנות מהגרף של פנינו:

- **איבודי הודעות:** בסימולציה הステטיסטית דרשנו 0 איבודים, ולכון התוצאה המנורמלת היא 1. יחד עם זאת, חלק מכובד מהתוצאות הדינמיות השיגו איבוד אפסי או קרוב אליו (תוצאה מנורמלת <0.99). התוצאות הגרועות ביותר ביוטר התקבלו עבור קצב דגימה איטי בשילוב עם threshold מסוימלי (כלומר - רק כאשר מגעים לתוך מלא מפעילים את כמות הרכיבים המקוריים), מה שגורם לעומס בתור ולאיבודים. קונפיגורציה זו דומה בהתנהוגתה למנגנון שמשימוש במערכת האמיתית ממנה לקחתי השראה, דבר שיכול להסביר את הביעות שמויפות שם (קרייסה של המערכת).
- **תפוקה:** הסימולציה הステטיסטית מגיעה לתפוקה נמוכה (0.37 לאחר נרמול) לעומת רוב הסימולציות הדינמיות, כאשר התוצאות הטובות ביותר ביוטר התקבלו עבור מערכות שמצוות על גבי משך זמן ארוך יותר (60 שניות) ועבור מערכות שגובהם הכיכי מהר לעליה וירידה (עדכון כל 0.25 15/30 שניות אחרונות). בקומבינציית בין המיצוע הוא על גבי 60 שניות, למערכת לוקחים זמן רב יותר להתאושש לאחר עלייה בכמות היחידות כתוצאה מעומס, וחולף זמן רב יותר עד להורדת של כמות הרכיבים. ניתן לראות שעבור מערכות אלו, זמן העבודה בפועל של הקטולוגים, שמתיחס רק בזמן בהם הרכיבים נמצאים בפועל נמוך משמעותית – ככלומר הרכיבים מוחזקים פעילים אך לא בעבודה.

- **השחיה ממוצעת ומקסימלית:** הסימולציה הステטיטית מביאה תוצאות טובות ממשמעותית מאשר בסימולציות הדינמיות שהרצינו, כאשר התוצאות הכיו טובות שמתקבלות בסימולציות הדינמיות הן עבור קצב העדכון הכי איטי (כל שנייה, ציון מנורמל של 0.3-0.32). לעומת זאת, הפערים בהשחיה המקסימלית קטנים משמעותית, ועבור הקומבינציות הניל' מתקבלת תוצאה (מנורמלת) של 0.81 לעומת 1 בסימולציה הステטיטית. המשמעות – ההשחיה הטובה ביותר ביותר מתקבלת עבור הסימולציה הステטיטית ללא תחרות אמיתית מהסימולציה הדינמית, אך במחיר גבוה גודל ברכיבים.
- **אורך תור ממוצע:** התוצאות הטובות ביותר מתקבלו עבור threshold נמוך (75), במיוחד כאשר משלבים זאת עם קצב העדכון הכי איטי (כל שנייה). התוצאות הגרועות ביותר מתקבלו עבור ה- threshold הכי גבוהה (100, אורך התור המקסימלי), בשילוב עם קצב הדגימה הכי מהיר (כל 0.25 שניות) – תוצאה זו מפתיעה, שכן היה ניתן לצפות שקצב עדכון מהיר יגרור שיפור באורך התור, אך בפועל נראה שבמצב זה המערכת אינה יציבה.
- **כמויות הקטלוגים פעילים:** זוהי המסה הגדולה של הרכיבים במערכת, ועבורם לקצב עדכון הייתה את ההשפעה הגדולה ביותר על התוצאות – ככל שקצב העדכון היה איטי יותר, כך הופעלו פחות רכיבים בממוצע. ניתן שזו נובע דווקא מאיכות התגובה, שלפעמים המערכת מספיקה להתחזון ולא מתකבל overshoot בכמות הרכיבים כתוצאה מתוגבה מהירה מדי.
- **כמויות הודעות בקרה:** כצפוי, עדכון בתדרות גבוהה יותר מוביל לעלייה בכמות הודעות בקרה. בנוסף, מיצוע על גבי זמן ארוך יותר גורר שליחת פחות הודעות בקרה (המערכת פחותת תנודתית במצב זה), אך השפעה זו זניחה לעומת השפעה של שינוי קצב העדכון.

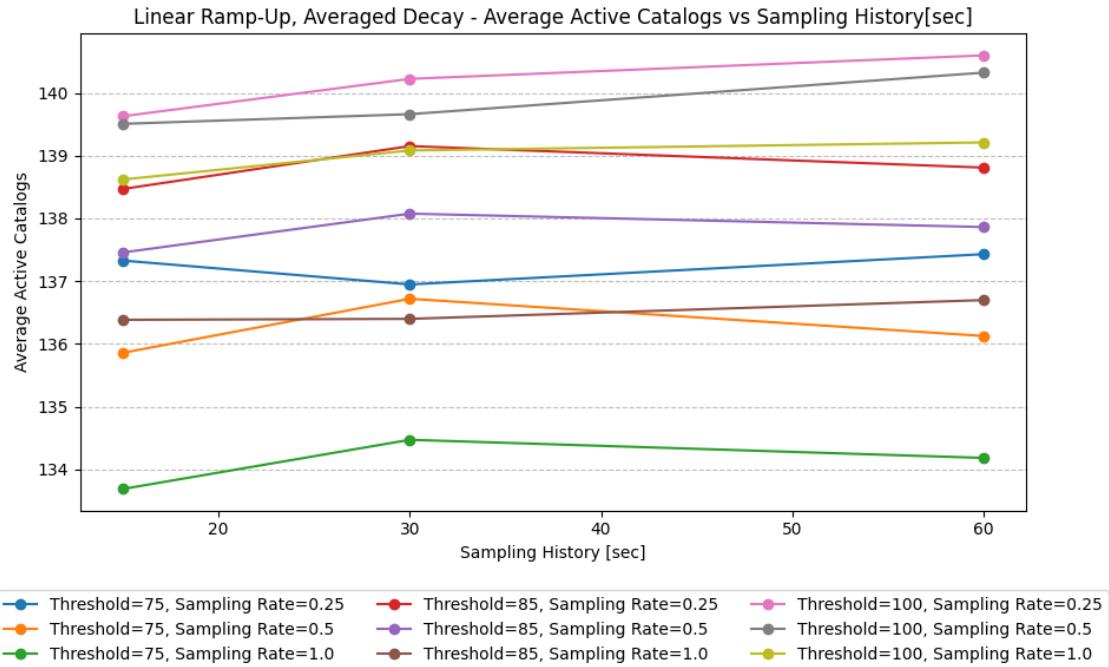


איור 10.7 : התפלגות התוצאות המנורמלות (כאשר 1 – התוצאה הכי טובה שנמדדה, 0 – הכי גרועה) לממדים שנבדקו, סימולציה דינמית (עליה לינארית, ירידה עפ"י היסטריזו).

גרף זה מציג את ההתפלגות של התוצאות המנורמלות, וממחיש עד כמה הן משתנות בין הקומבינציות השונות של הפרמטרים שבדקנו, וסביר אילו ערכיהם הם מתרכזות, דבר שמציג את מידת ההשפעה של הפרמטרים שנבדקו על כל ממד.

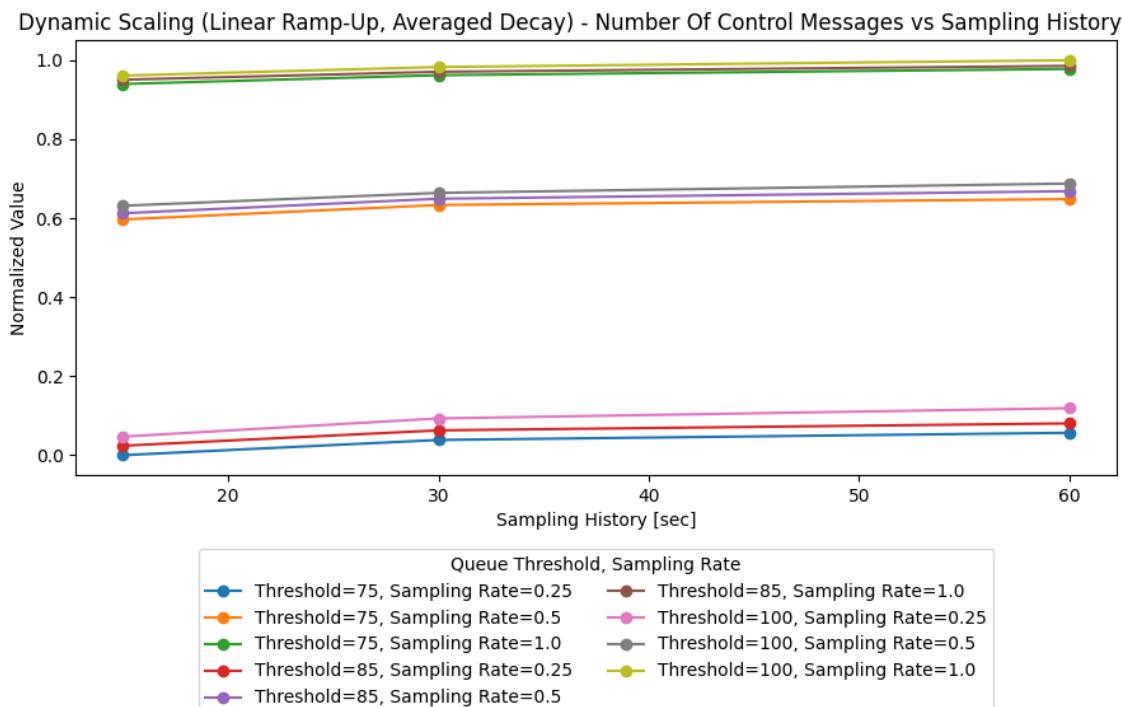
ניתן לחלק את הממדים למספר קבוצות:

- **טוח ערכים רחב** (כמויות הודעות הבקרה, אורך טור ממוצע, תפוקה) : ממדים אלו מושפעים בצורה מהותית ע"י הפרמטרים שבדקנו, ומציגים פיזור רחוב בתוצאות.
- **טוח ערכים צר** (זמן פעולה בפועל, השהייה מקסימלית) : ממדים אלו פחות הושפעו מהפרמטרים שבדקנו, ופונקציית הבקרה הלינארית מספקת לרובם תוצאות דומות למדי.
- **ממוצע גובה** (מדד droprate) : ברוב הקומבינציות התוצאות שהתקבלו היו טובות מאוד, ניתן להסיק שפונקציית הבקרה היא הגורם לתוצאות אלו, ולא בחירת הפרמטרים (מלבד בחירת פרמטרים ספציפית חריגה שהניבת תוצאות גרועות).
- **ממוצע נמוך** (כמויות קטלוגים פעילה) : כאן, מלבד בחירת נקודתית של פרמטרים הרוב המוחלט של התוצאות היו לא טובות. ככל הנראה בגלל מגבלת של פונקציית הבקרה עצמה. יחד עם זאת, בחירה חכמתה של פרמטרים כן יכולה להניב תוצאה טובה.



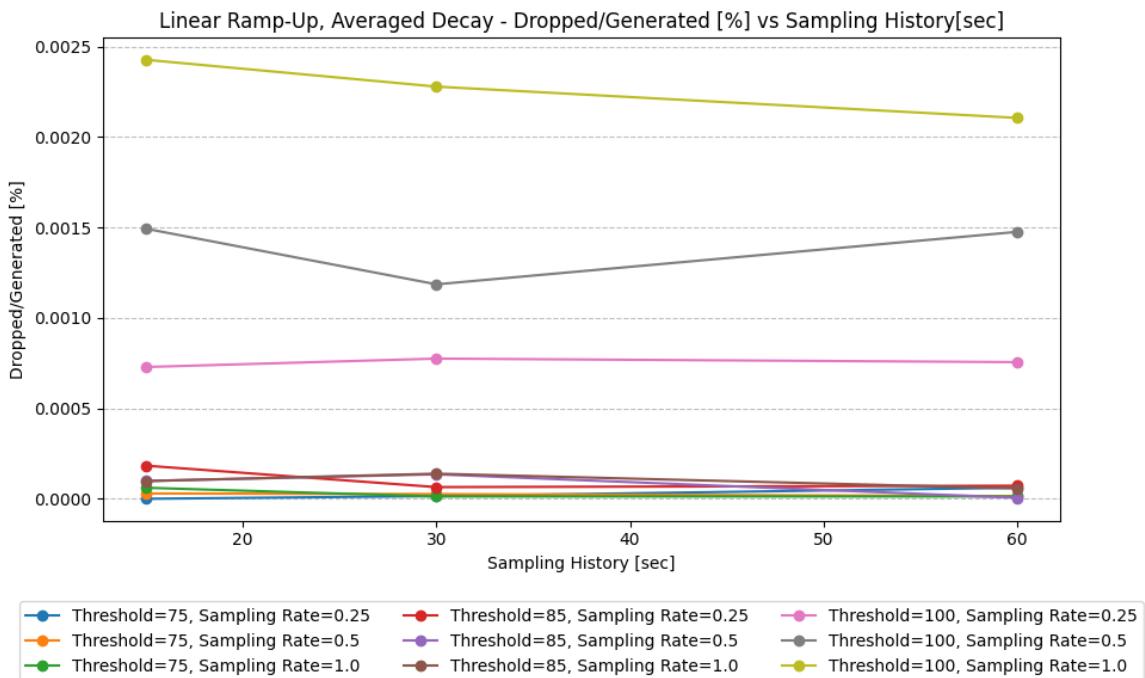
אייר 7.11 : כמות הקטלוגים הפעילה בממוצע כפונקציה של היסטוריות הדגימות (בשניות), עבור קומבינציות של קבועי דגימה ו-threshold שונים, סימני דינמיות (עליה לינארית, ירידת עפ"י היסטורייז).

ניתן להבחן כי עם הפחתת ה-threshold, כמות הקטלוגים קטנה – המערכת מגיבה "זמן" לעומס, ולכן נמצאת פחות זמן עם כמות רכיבים פעילים גבוהה, בדgesch על כאשר קבוע הדגימה הינו האיטי ביותר – דבר שימושי לטובתנו פה.



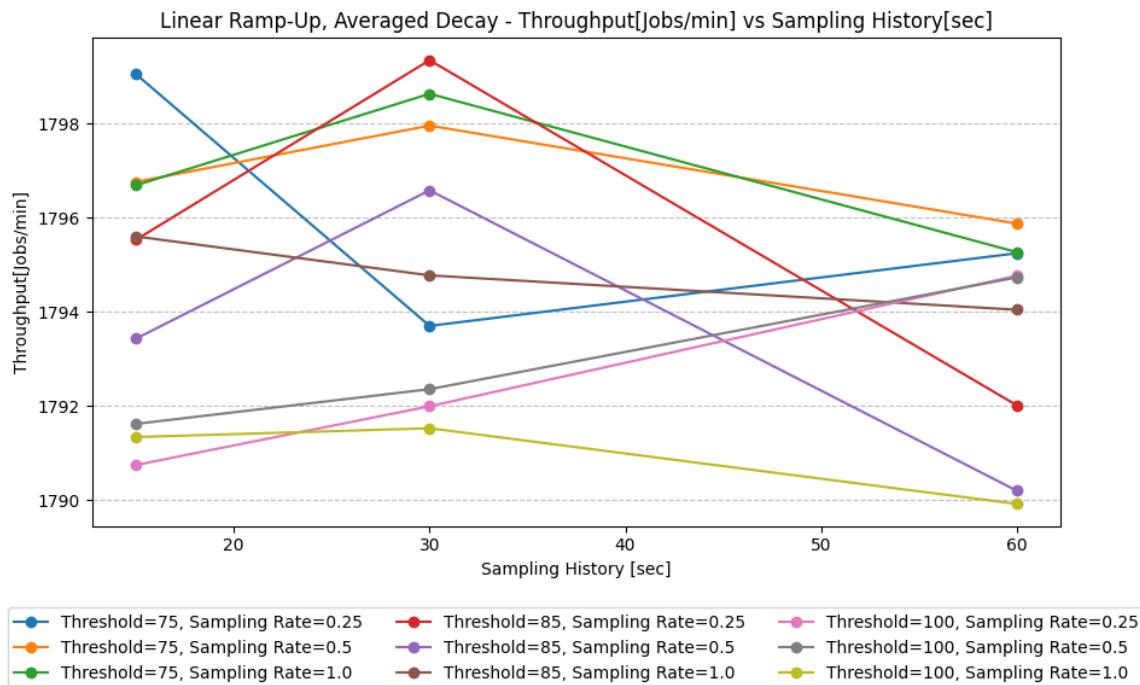
אייר 7.12 : כמות הודעות הבקרה המנורמלת (גובה יותר =פחות הודעות) כפונקציה של היסטוריות הדגימות (בשניות), עבור קומבינציות של קבועי דגימה ו-threshold שונים, סימני דינמיות (עליה לינארית, ירידת עפ"י היסטורייז).

אפשר לראות קשר חזק בין קבוע הדגימה לבין כמות הודעות הבקרה – עבור קבוע הדגימה איטי נשלחות פחות הודעות בקרה מאשר עבור קבוע דגימה מהיר יותר, וכך מעדכו מהיר יותר של המערכת לשינויים.



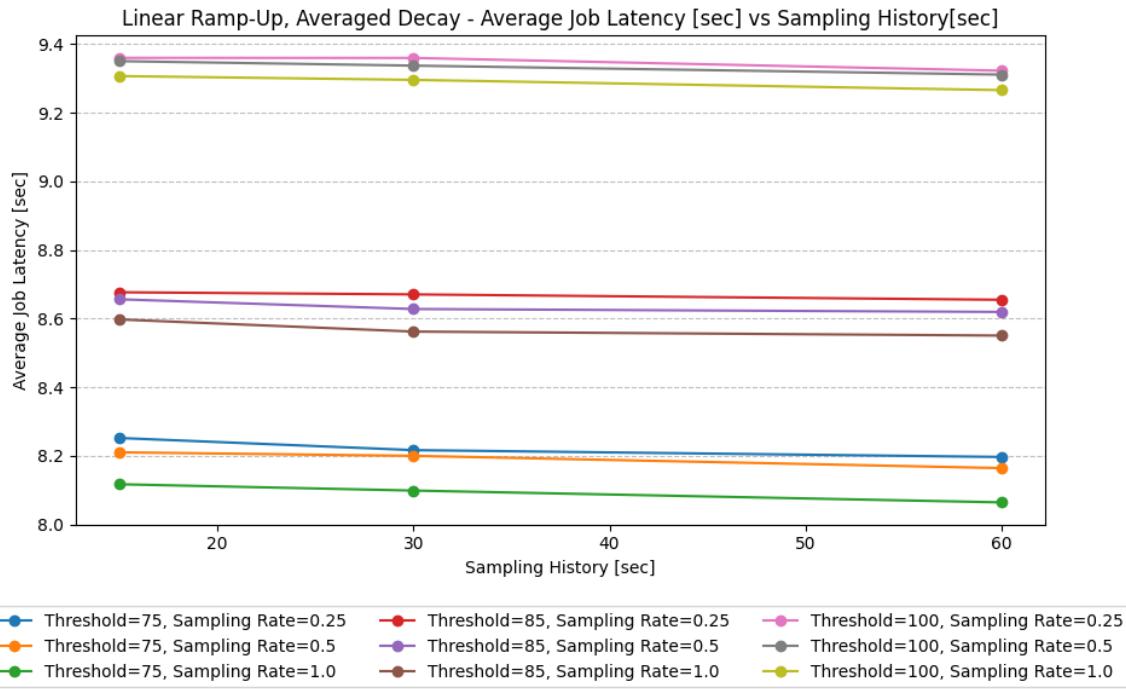
אייר 13.7 : אחוז הטעות שנפלו מסך הטעות כפונקציה של היסטוריית הדגימות (בשניות), עבור קומבינציות של קבועי דגימה ו-threshold שונים, סימן דינמית (עליה לינארית, ירידה עפ"י היסטורייז).

כאן ניתן לראות בבירור שעבור threshold מקסימלי אנחנו מקבלים משמעותית יותר איבוד הדעות, כיון שבקונפיגורציות אלו המערכת מגובה "מאוחר מדי" לעומס המגע.



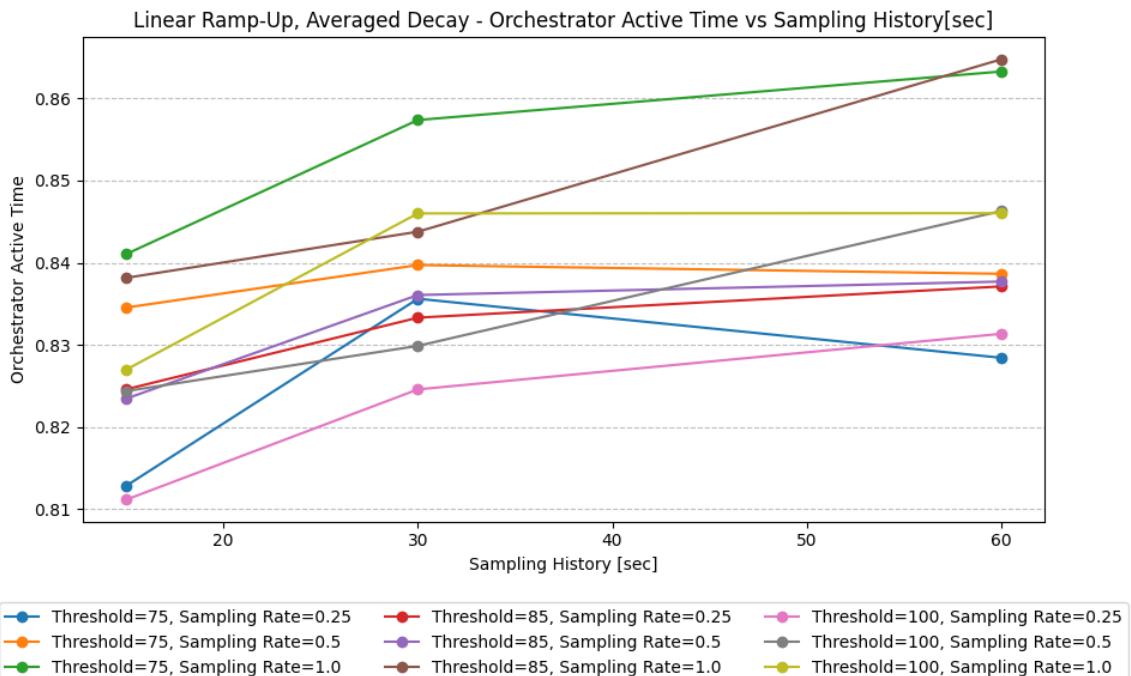
אייר 14.7 : תפקה (טעות לדקה) כפונקציה של היסטוריית הדגימות (בשניות), עבור קומבינציות של קבועי דגימה ו-threshold שונים, סימן דינמית (עליה לינארית, ירידה עפ"י היסטורייז).

התפקה מושפעת ממספר פרמטרים, אך קשה לראות יתרון מובהק לאחת מהקומבינציות שבדקו. יחד עם זאת, התפקה הטובה ביותר לרוב המוחלט של הקומבינציות היה עבור היסטוריית דגימה של 30 שניות.



אייר 7.15 : ההשיה הממוצעת לעובדה כפונקציה של היסטוריית הדגימות (בשניות), עבור קומבינציות של קבוע threshold ו-sampling rate שונים, סימן דינמי (עליה לינארית, ירידה עפ"י היסטוריון).

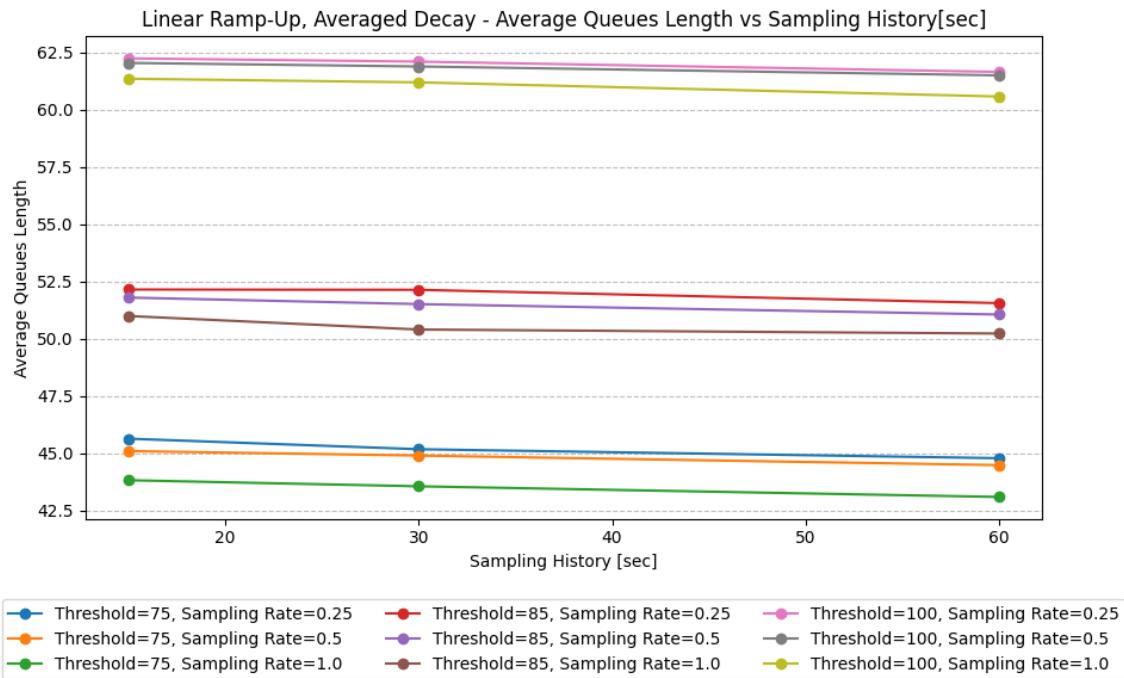
עבור threshold קטן יותר, אנחנו מקבלים השהייה נמוכה מדי – המערכת מגיבה מהר יותר לעומס, ועל כן מפעילה עוד ייחדות מהר יותר בצדיה להתמודד.



אייר 7.16 : שיעור זמן העבודה הממוצע בפועל של המתאים כפונקציה של היסטוריית הדגימות (בשניות), עבור קומבינציות של קבוע threshold ו-sampling rate שונים, סימן דינמי (עליה לינארית, ירידה עפ"י היסטוריון).

ניתן לראות קשר ישיר בין שיעור השימוש בפועל (כמה זמן הם באמת היו פעילים) של הרכיבים לאורך ההיסטוריה – מיצוע עיג ההיסטוריה ארוכה יותר גורם למערכת להגיב לאט יותר, ועל כן יותר רכיבים או

כבויים או לא בעבודה. ניתן לראות גם שימוש גבוה יותר ברכיבים בפועל עבור קצב הדגימות האיטי, מסיבה דומה של זמן תגובה.



אייר 2.7 : אורך התורים הממוצע במערכת כפונקציה של היסטורית הדגימות (בשניות), עבור קומבינציות של קבועים ו-threshold שונים, סימן דינמי (עליה ליארית, ירידה עפ"י היסטורי).

תוצאות אלו מספקות תובנה מעניינת – זמן עבודה בפועל גבוה איןו מוגרם בהכרח לתורים קצרים יותר – כפי שניתן לראות עבור threshold 100. ההשפעה המרכזית היא דוקא בחירת ה-threshold.

7.1.3. מציאת הקונפיגורציה האופטימלית

נשותמש בפונקציית הציון שהגדכנו קודם לכן, ונחשב את הציון של כל אחת מן הקומבינציות של הפרמטרים שבדקנו:

Sampling Rate [1/sec]	Queue Threshold	Sampling History [sec]	Score [0-3.2]
0.25	100	15	0.82567
		30	0.78747
		60	0.80434
	75	15	2.32553
		30	2.41346
		60	2.18523
	85	15	1.76359
		30	1.67745
		60	1.70576
0.5	100	15	0.70852
		30	0.88243
		60	0.78828
	75	15	2.43532
		30	2.36243
		60	2.46018
	85	15	1.8821
		30	1.84806
		60	1.87871
1	100	15	0.75388
		30	0.635
		60	0.79061
	75	15	2.81545
		30	2.70246
		60	2.63985
	85	15	2.01514
		30	2.15394
		60	1.87657

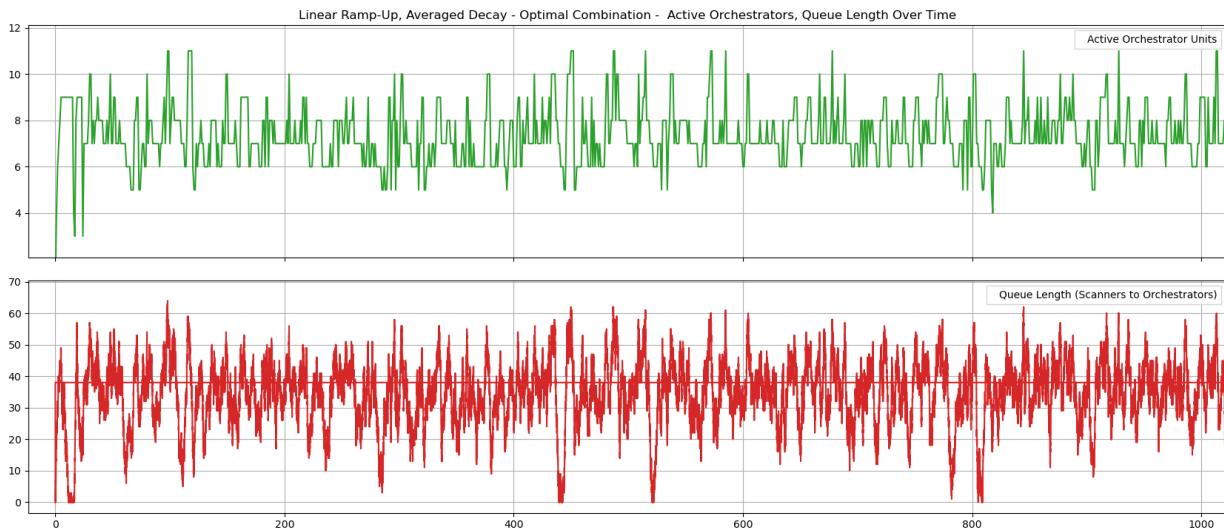
טבלה 7.1: טבלת הציוןים לכל הקומבינציות של הפרמטרים שנבדקו.

עבור המשקלות שהגדכנו, יש העדפה ברורה **לקצב דגימה איטי יותר** (של דגימה אחת לשנייה) בשילוב עם **סף של 75** (הפעלה של כל הרכיבים כאשר התורים מגיעים ל-75 הודעות/עבודות). כאשר הזמן אותו אנחנו ממצאים ארוך יותר, קצב התגובה של המערכת (עבור ה-scale-down) נמוך יותר, מה שגורר עלייה בכמות הרכיבים המומוצעת.

לכן בסה"כ הקונפיגורציה האופטימלית מבין אלו שנבדקו היא עם קצב דגימה של דגימה אחת לשנייה, סף מקסימלי לתורים של 75 הודעות/עבודות ומיצוע על בסיס 15 השניות האחרונות.

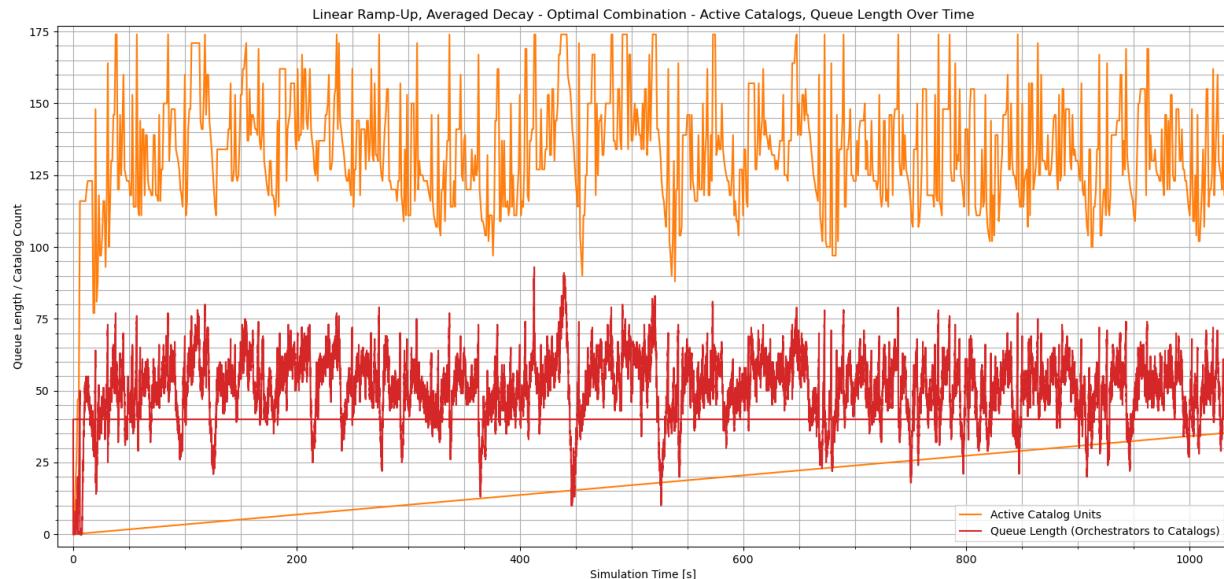
7.1.4. הkonfigורציה האופטימלית – מדדים לאורך זמן

הkonfiguraciah оптимальна: קצב דגימה – 1 לשניה, ספ' מקסימלי לטוררים – 75 הודעות, מיצוע ע"ב 15 שניות אחרות.



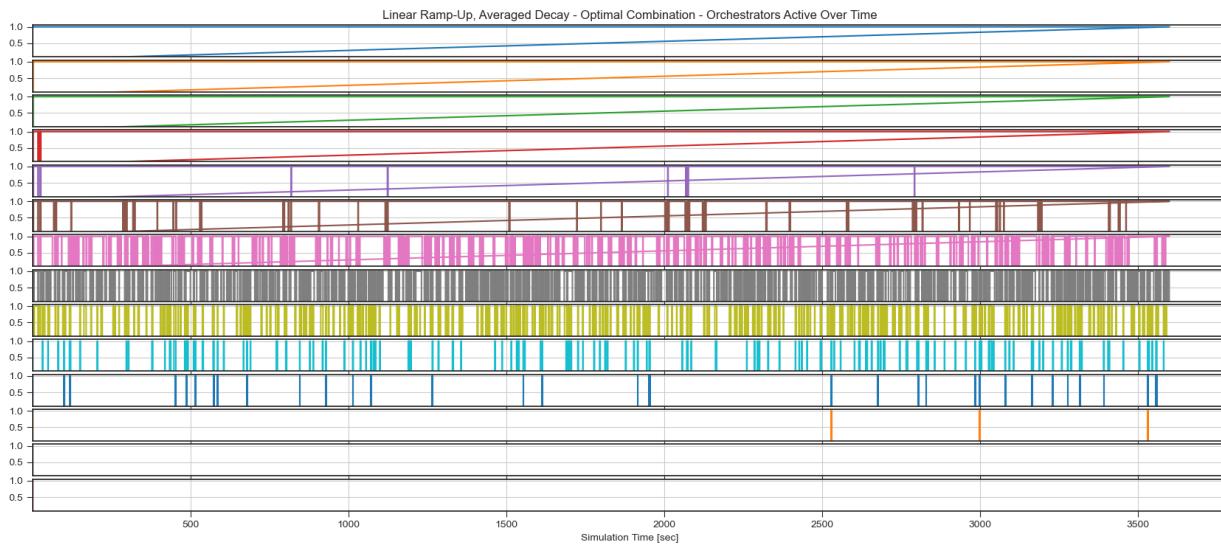
אייר 18.7: כמות המתאימים הפעילים ואורך התור המזין אותם לאורץ זמן עבור הפרטורים האופטימליים שנבחרו, סימן דינמיית (עליה לינארית, ירידתית עפ"י היסטורי).

אורך התווך הממוצע תנודתי במידה מסוימת, אך ברובו נשאר בין 50-30, הרחק מהאורך המקורי. אפשר לראות שכאשר אורך התווך עולה, כמו הרכיבים עולה בעקבותיו, ולאחר מכן יורדת בחלוף הזמן.



אייר 19.7: כמות הקטלוגים הפעילים ואורך התור המזון אותם לאורך הזמן עבור הפרטוריים האופטימליים שנבחרו, סימני דינמיות (עליה לינארית, ירידיה עפ"י היסטורי).

בדומה לאורך התווך במקורה של המתאים, גם כאן ניתן לראות את התגובה בהעלאת כמות הרכיבים כאשר אורך התווך גדול, ולאחר מכן מתאזן גם את הירידה בכמות הרכיבים הפעילים. כמוות הרכיבים הפעילים מגיעה למקסימום (174) מספר רב של פעמים במהלך הסימולציה, מה שמרמז על צורך בהעלאת התקראה המקסימלית כדי לאפשר לפונקציית הבקרה לעבוד באופן מיטבי.



איור 7.20 : שיעור זמן העבודה בפועל של המתאים לאורך זמן עברו הפרמטרים האופטימליים שנבחרו, סימי דינמיות (עליה לינארית, ירידה עפ"י היסטריז).

אפשר לראות כי במהלך כל הסימולציה, מתאים 12 ו-13 אינם מופעלים כלל, בעוד שמתאים #0 עד #3 מופעלים במהלך כל הסימולציה. מתאים 9-5 פעילים לסירוגין במהלך הסימולציה.

2.7. סקר ביצועים – שיעור זמן לריקון התור (Queue Drain-time Estimation)

בין החסרונות הבולטים של המנגנון הלינארי שבדקנו קודם הם ההשניה והתפוקה, שרחוקים מאוד מהפתרונות עבור הסימולציה הステטיסטית.

כדי להתמודד עם זה בצורה טובה יותר, נטמע ונבדוק מנגנון בקרה שם דגש על ההשניה ותפוקה ע"י ניסיון לשערן את ההשניה הצפואה של המערכת, ומגדיל או מקטין את כמות הרכיבים כך להגיע להשניה הרצוייה.

המנגנון עובד באופן הבא:

- כל X שניות, מתבצעת מדידה של אורך התורים ומחושב קצב הריקון המשוער באופן הבא :

$$\text{Drain Rate} = \frac{\text{Queue Length}}{(\# \text{ of Current Pods}) \cdot (\text{Service Rate})}$$

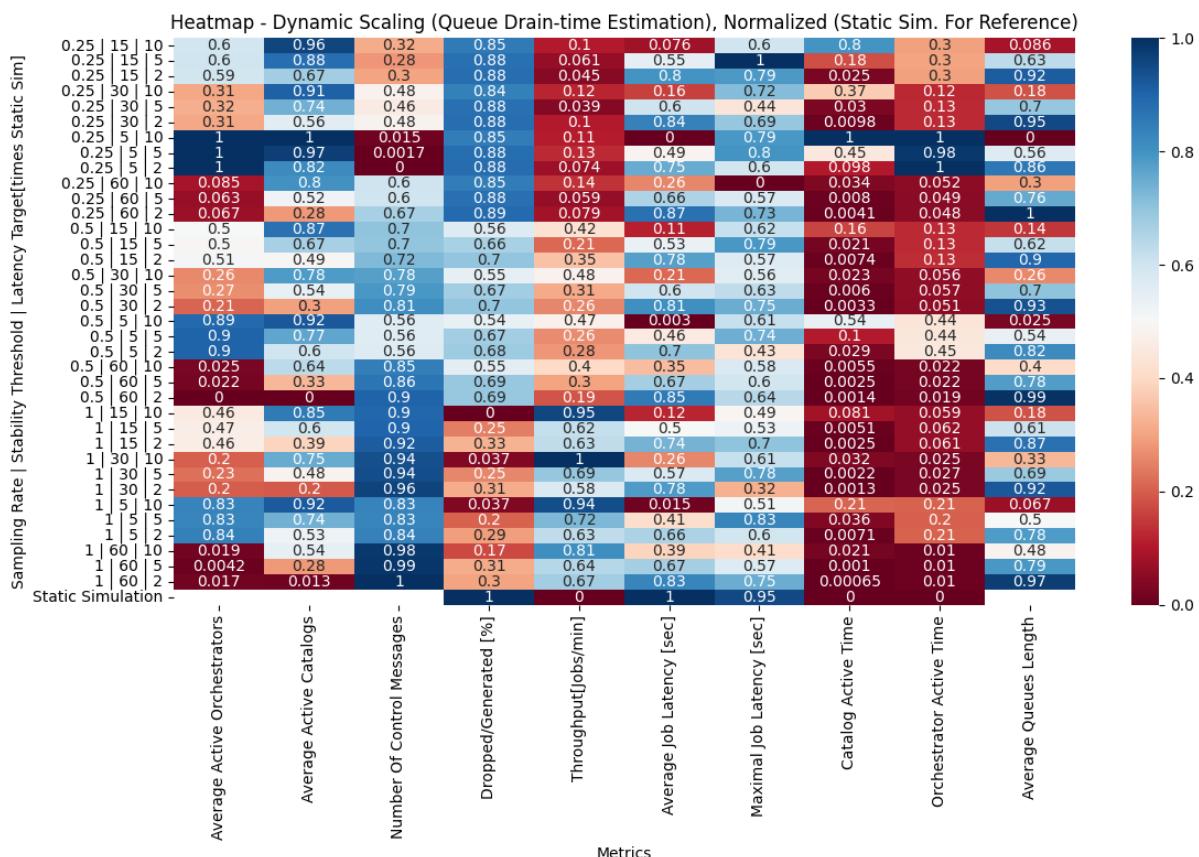
- אם $\text{Drain Rate} > \text{Target Latency}$: מפעילים עוד יחידה.
- אם $\text{Drain Rate} < \text{Target Latency}$: מבאים יחידה.

באופן זהה, ניתן לבצע חזרה מבודקת של יחידות באופן דומה למיצוע המתבצע בפונקציית הבדיקה הלינארית, כאשר העלאה מתבצעת באופן דינמי יותר.

מבצע סקר ביצועים על הפרמטרים הבאים:

- כמות פודים מינימלית : 1 (עבור מתאימים וקטלוגים).
- כמות פודים מקסימלית : עבור מתאימים – 14, עבור קטלוגים – 174. (מהסימולציה הステטיסטית).
- משך הזמן בו המערכת במצב יציב (ז) : 60/15/30/60 5/15/30/60 שניות.
- ערך היעד של ההשניה – ביחס להשניה של כל אחד מהتورים בסימולציה הステטיסטית : פי 2/10/5.
- קצב הדגימה (כל כמה זמן דוגמים את אורך התורים) : 1, 0.5, 0.25, 0.1 שניות.

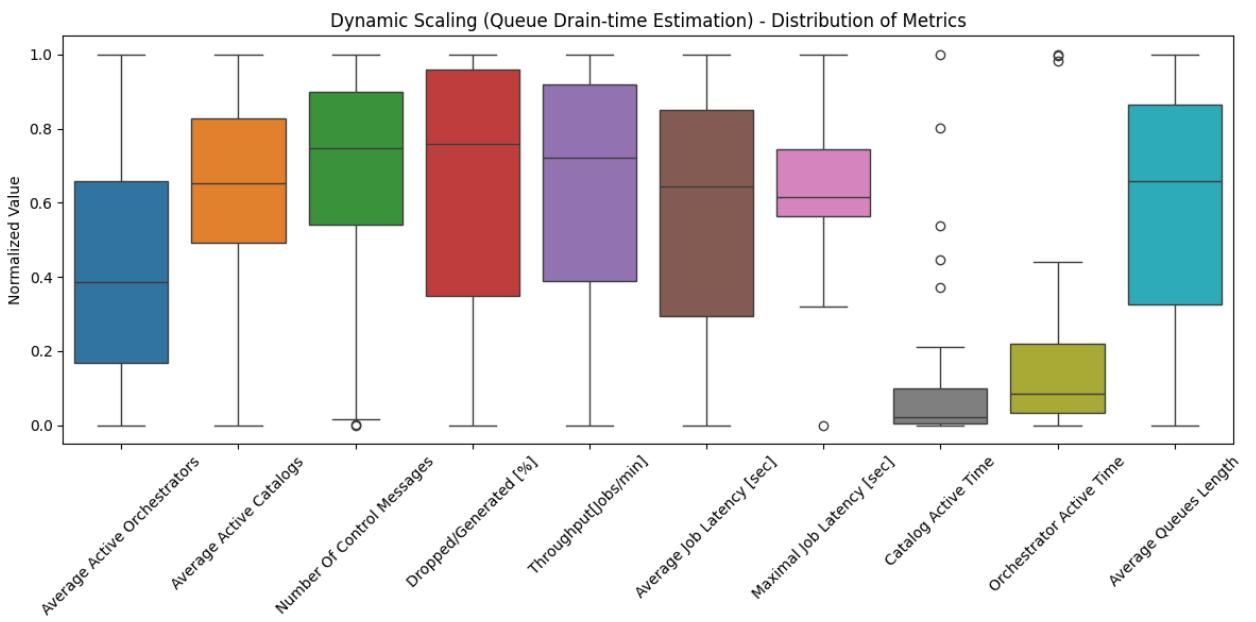
7.2.1. תוצאות כוללות – Multiple Sweeps



אייר 7.21: Heatmap מנורמל לסדרת הפרמטרים עבור הסימני הדינמי (שיעורן זמן לריקון התוור - QDTE), ביחס לסימולציה הսטטistica (שורה תחתונה).

כמה תובנות מהגרף:

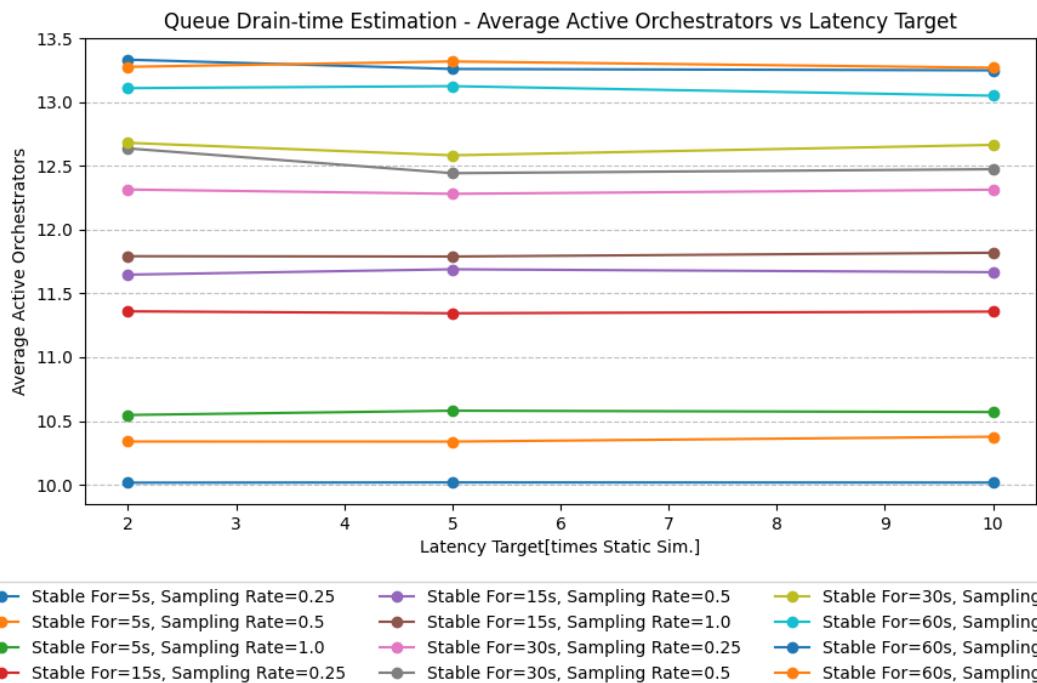
- כל הקומבינציות שנבדקו השיגו תפוקה גבוהה יותר לעומת התפוקה הסטטיסטית, כאשר התפוקות הגבוהות ביותר התקבלו עבור קצב הדגימה האיטי ביותר (כל שנייה אחת).
- כל הקומבינציות שנבדקו איבדו הודעות בכמות צו או אחרת, כאשר שיעור האיבודים הכי נמוכים התקבלו עבור קצב הדגימה הכפי מהיר (כל 0.25 שניות). אפשר לראות כי ישיחס הפוך בין התפוקה לשיעור איבוד ההודעות.
- כפי שנῆפה מפונקציית הבקרה, אורך התורדים וכמות היחידות הפועלות מושפע בצורה ניכרת מערך היעד של ההשניה – ככל שההשניה הרצiosa נמוכה יותר, כך אורך התוור הממוצע נמוך יותר וכמות היחידות הפעילות גבוהה יותר.
- כמוות הודעות הבקרה שנשלחות מושפע כפוי מקצב הדגימה (קצב דגימה איטי יותר גורר פרות הבודעות בקרה), אך נוכל לשים לב שיש השפעה לא מבוטלת גם משך הזמן לציבות, כאשר הוא גבוה יותר כמות הודעות הבקרה קטנה, ואילו כאשר הזמן לציבות קצר יותר המערכת יותר.
- כמו כן, ההשניה הממוצעת הניתנת ביוטר מתקבלת עבור זמן היציבות הארוך ביותר בשילוב עם זמן התגובה הכפי מהיר וערך היעד יותר – תוצאה זו נובעת מכך שבמצב זה המנגנון לא ממהר להפחית את כמות היחידות, ועל כן גם מחזיק בכמות היחידות דלוקות ממוצעת גבוהה.



אייר 7.22 : התפלגות התוצאות המנורמלות למדדים שנבדקו, סימולציה דינמית (שיעורן זמן לריקון התו"ר - QDTE).

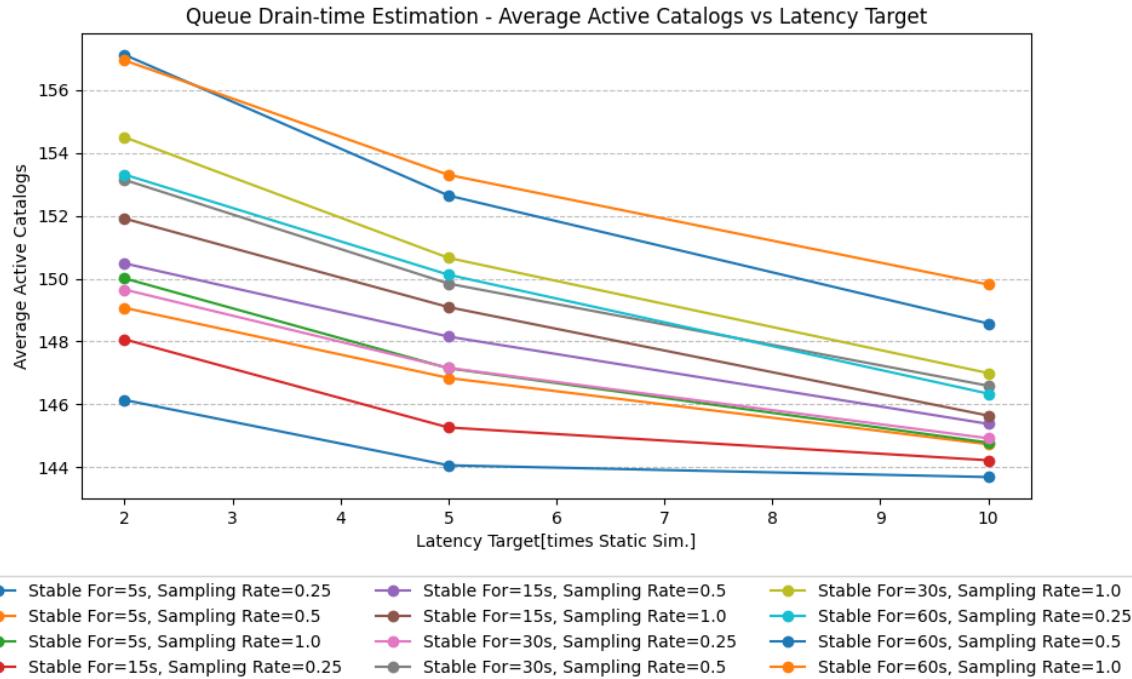
ניתן להבחין שההתפלגות של התוצאות (הציוונים המנורמלים) רחבה למדי ברוב הקטגוריות, כאשר יש הפרשים גדולים בין התוצאות השונות עבור יחס איבודי הידועות, **כמויות המתאמים הפעילים, אורכי התו"רים וההשיה הממוצעת** (כמצופה, כיוון שבדקנו מס' ערכי יעד להשיה).

- לעומת זאת, עבור ההשיה המקסימלית מתקבלות תוצאות קרובות למדי עבור מרבית הקומבינציות – ככלומר, על אף שינויים גדולים בהשיה הממוצעת, הערכים המקסימליים דומים למדי בין רובם.



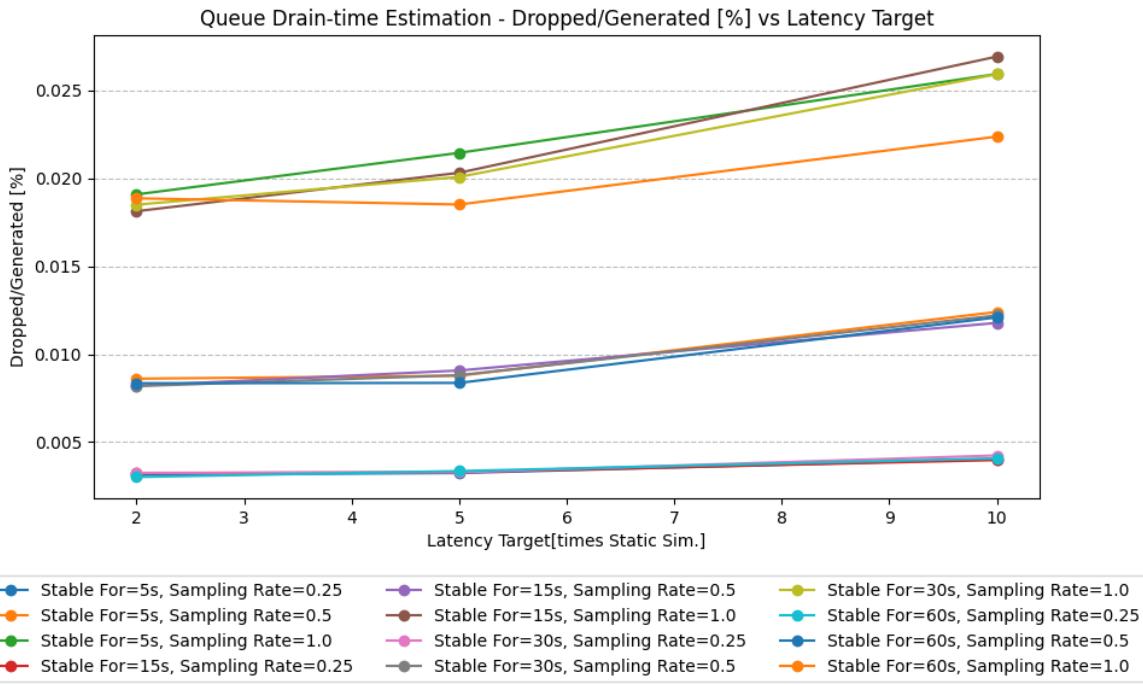
אייר 7.23 : **כמויות המתאמים הפעילים בממוצע כפונקציה של יעד ההשיה, עבור קומבינציות של קבועי דגימה וזמן יציבות שונים, סימוי דינמית (שיעורן זמן לריקון התו"ר - QDTE).**

אפשר לשים לב שכבות המתאמים (Orchestrators) הפעילים אינו תלוי ביעד ההשניה. אפשר לראות כי מدد זה מושפע כמעט לחלוטין בלבד ע"י זמן יציבות – זמן יציבות של 60 שניות גורר כמות ממוצעת גבוהה, ואילו זמן יציבות נמוך (5 שניות) מאפשר למערכת להגיב בצורה יותר דינמית לשינויים בעומס, ועל כן להקטין את כמות הרכיבים הפעילים. **גרף זה מעיד על כך שניתן להגעה יתר-קלות ליעד ההשניה עבור התווך הראשון** (אין כמעט השפעה על כמות היחידות הפעילות כאשר מחמירים את היעד).



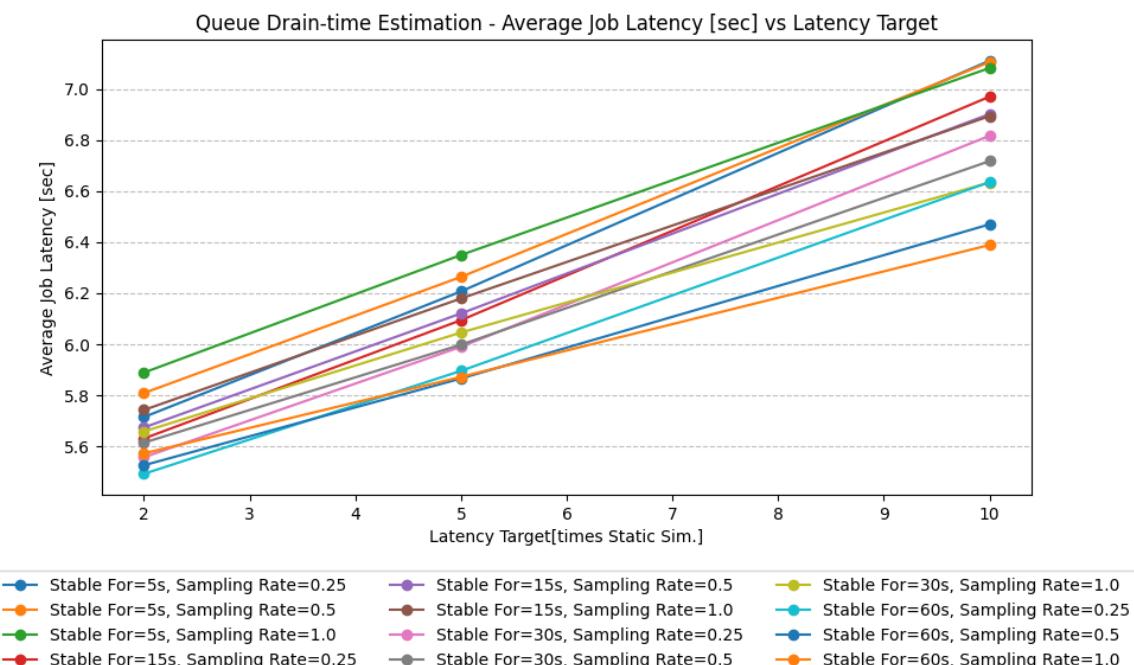
אייר 7.24 : כמות הקטלוגים הפעילים בממוצע כפונקציה של יעד ההשניה, עבור קומבינציות של קבוע דגימה וזמן יציבות שונים, סימני דינמית (שיעור זמן לריקון התווך - QDTE).

כמות הקטלוגים במערכת מושפעת בצורה ניכרת ע"י יעד ההשניה – מה שמעיד על כך שתווך זה הינו צואր הבקבוק של המערכת. גם כאן, זמן יציבות גבוהה יותר גורר כמות ממוצעת גבוהה יותר, אך בניגוד לכמות המתאים הפעילים, כאן יש השפעה לעד ההשניה על הכמות הממוצעת.



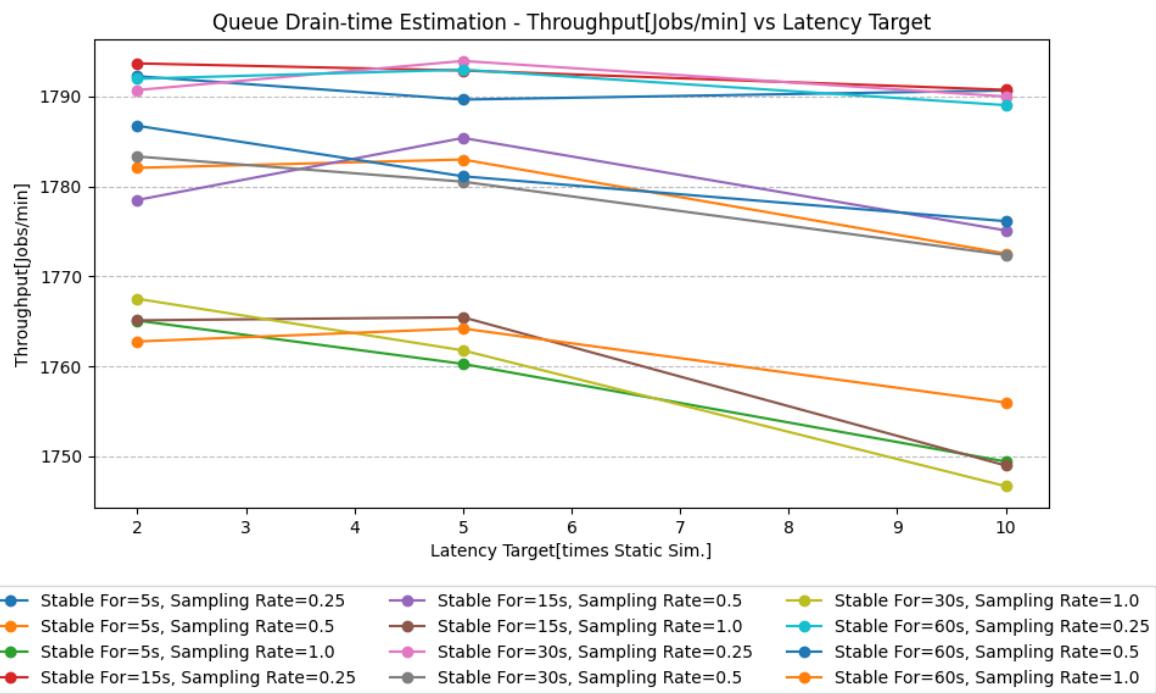
אייר 7.25 : אחוּ העבודות שנפלו מסך העבודות כפונקציה של יעד ההשאה, עבור קומבינציות של קבועי דוגמה וזמן יציבות שונים, סימני דינמית (שיעור זמן לרכיבון התוֹר - QDTE).

ניתן לראות כי אחוּ איבודי העבודות במערכת מושפע בעיקר מזמן התגובה הכלול של פונקציית הבקרה – עבור זמן יציבות גבוהים יותר, שיעור האיבוד נמוך יותר. עבור זמן יציבות נמוך (5 שניות) קיבלנו את שיעור האיבוד הגבוה ביותר. במצב זה, המערכת דינמית מדי ומתקשה להגיב בזמן לעלייה בעומס, על כן מגעווים למצב שבו התורים מלאים.



אייר 7.26 : ההשאה הממוצעת לעובדה כפונקציה של יעד ההשאה, עבור קומבינציות של קבועי דוגמה וזמן יציבות שונים, סימני דינמית (שיעור זמן לרכיבון התוֹר - QDTE).

כפוי, פונקציית הבקרה מאפשרת לנו לעקוב אחר ההשאה הרצויה, ונitin' לראות בבירור כי ככל שנחמיר את הדרישה על ההשאה, המערכת מגיבה בהתאם.



אייר 7.27 : תפקה (עבודות לדקה) כפונקציה של יעד ההשניה, עבור קומבינציות של קצב דגימה וזמן יציבות שונים, סימני דינמיות (שיעורן זמן לרכיבון התור - QDTE).

ניתן להבחן שהתפקה הכלולת של המערכת מושפעת בעיקר מקצב הדגימה (הערכים הטובים ביותר מתקבלים עבור קצב דגימה של 4 דגימות לשנייה). כאמור, כאשר ערך היעד עבור ההשניה נמוך יותר, התפקה יורדת בהתאם.

7.2.2. מציאת הקונפיגורציה האופטימלית

נשותמש בפונקציית הציון שהגדכנו קודם לכן, ונחשב את הציון של כל אחת מהתוצאות שנבדקו:

Sampling Rate [1/sec]	Stable For – [sec]	Latency Target [# times Static Sim.]	Score
0.25	5	2	2.602653
		5	2.341793
		10	1.681344
	15	2	2.510218
		5	2.306525
		10	1.60218
	30	2	2.371318
		5	2.109467
		10	1.602517
	60	2	2.227707
		5	1.998911
		10	1.458868
0.5	5	2	2.363285
		5	2.172426
		10	1.497744
	15	2	2.264634
		5	2.077314
		10	1.484618
	30	2	2.14515
		5	1.962808
		10	1.442465
	60	2	1.976572
		5	1.853262
		10	1.47656
1	5	2	2.120517
		5	1.871398
		10	1.244414
	15	2	2.023003
		5	1.741417
		10	1.160883
	30	2	1.798693
		5	1.705741
		10	1.209273
	60	2	1.746612
		5	1.626816
		10	1.256729

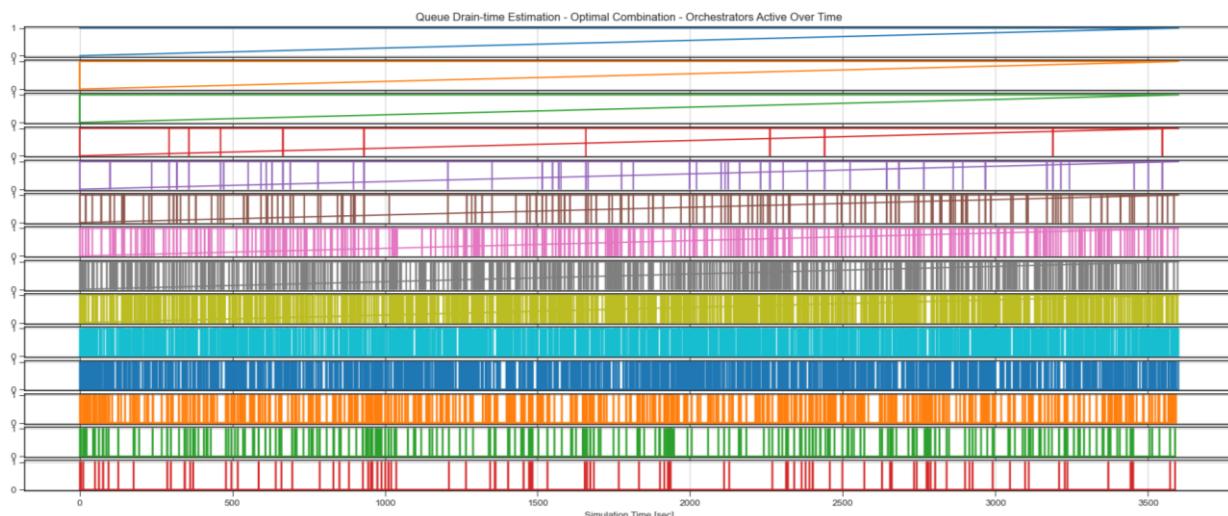
טבלה 7.2 : טבלת הציון לכולם הקומבינציות של הפרמטרים שנבדקו, color-coded מהטוב (ירוק) לגרוע (אדום)

ניתן לראות העדפה בולטת לקצב דגימה מהיר יותר (של 4 דגימות לשניה) בשילוב עם יעד ההשאה המהמיר ביותר (פי 2 מערך בסימולציה הסטטית), כאשר זמן היציבות פחות השפיע על הזמן הסופי (שלושת הקומבינציות הטובות ביותר ביותר הן עבר יציבות של 5,15 או 30 שניות).

בsha"ב, הזמן הטוב ביותר (2.602) התקבל עבור בחירת הפרמטרים: קצב דגימה של 0.25 שניות/דגימה, זמן יציבות של 5 שניות, ויעד ההשאה המהמיר ביותר – פי 2 מההשאה בסימולציה הסטטית.

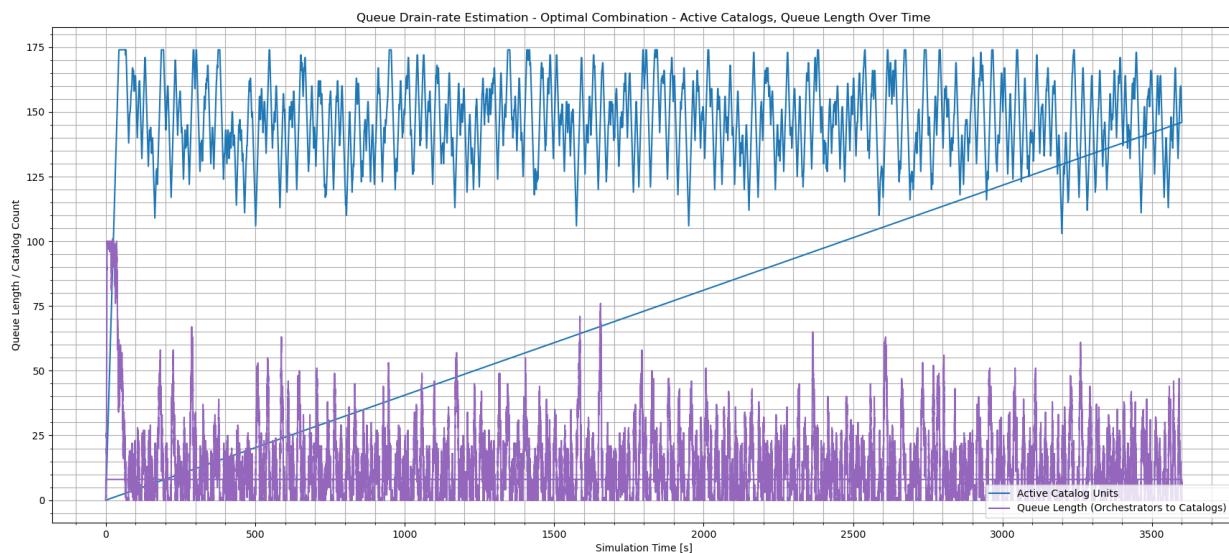
7.2.3. הקונפיגורציה האופטימלית – מדדים לאורך זמן

הkonfiguracija האופטימלית: קצב דגימה – 0.25 שניות לדגימה, זמן יציבות של 5 שניות (לפני הורדה), פי 2 מההשאה בסימולציה הסטטית. latency target



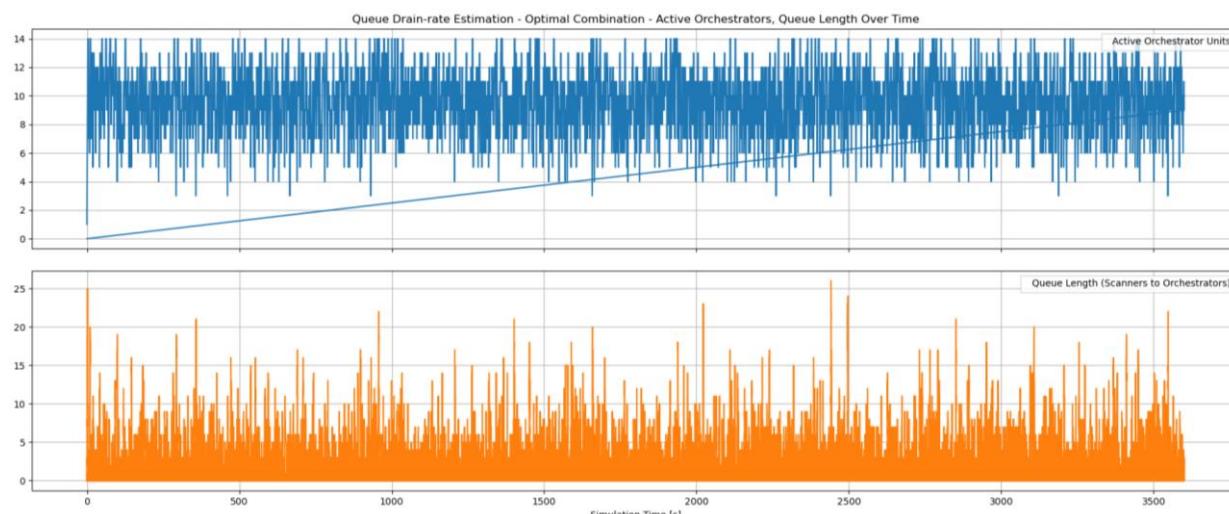
אייר 7.28 : שיעור זמן העבודה בפועל של המתאים לאורך זמן עבור הפרמטרים האופטימליים שנבחרו, סימ' דינמית (שיעורך זמן לריקון התור – QDTE).

ניתן לראות כל המתאים (מלבד האחרון) עובדים ברוב המקרה של הסימולציה כדי לעמוד ביעד ההשאה – אנחנו משלמים בהפעלת יותר רכיבים בכך להשיג השאה נמוכה יותר.



אייר 29. : כמות הקטלוגים הפעילים ואורך התור המזין אותם לאורך זמן עברו הפרמטרים האופטימליים שנבחרו, סימני דינמיות (שיעורן זמן לריקון התור – QDTE).

מלבד 'פיק' נקודתי עם תחילת הסימולציה (שמשם נובעת כל איבודי ההודעות), בו לפונקציית הבקרה לוקח זמן 'להתניע' בצדיו להתמודד עם העומס הנוצר (היות והיא מתחילה מכמויות היחידות המינימלית), פונקציית הבקרה שומרת על אורך תור ממוצע יציב יחסית ורחוק מהאורך המקסימלי (100). ניתן לראות כי אין עיכוב בהורדת כמות היחידות לאחר עומס, ובאופן ייחסי מהיר כמות היחידות הפעילות יורדת בהתאם לצפי ההשניה הנוכחי.



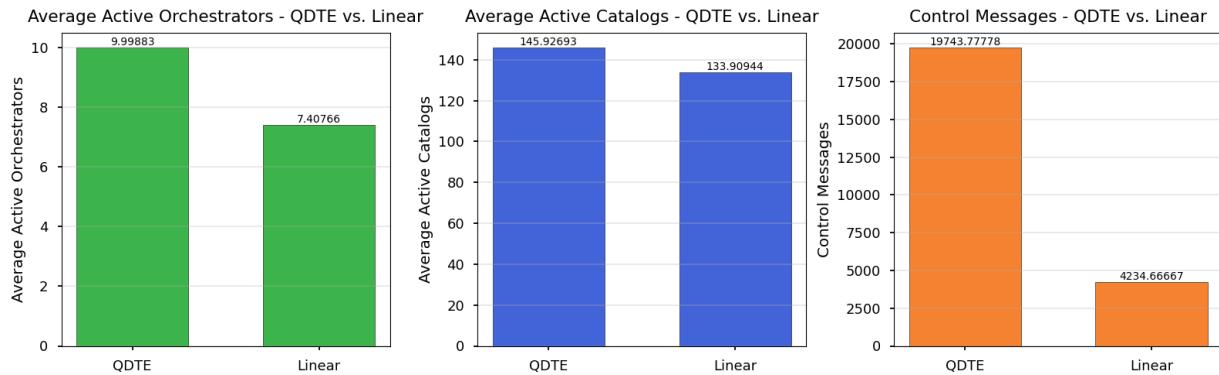
אייר 30. : כמות המתאיםים הפעילים ואורך התור המזין אותם לאורך זמן עברו הפרמטרים האופטימליים שנבחרו, סימני דינמיות (שיעורן זמן לריקון התור – QDTE).

בניגוד לפונקציית הבקרה הלינארית אותה בדקנו קודם, כמות המתאיםים הפעילים הממוצע גבוהה יותר, וזאת כיון שיעד ההשניה בו התור נדרש לעמוד הינו מחמיר יותר מאשר הדרישה בפונקציית הבקרה הלינארית, שהיא שליטה באורך התור ללא חתיכות מיוחדת להשניה.

3.7. סקר ביצועים – השוואת בין המנגנונים השונים

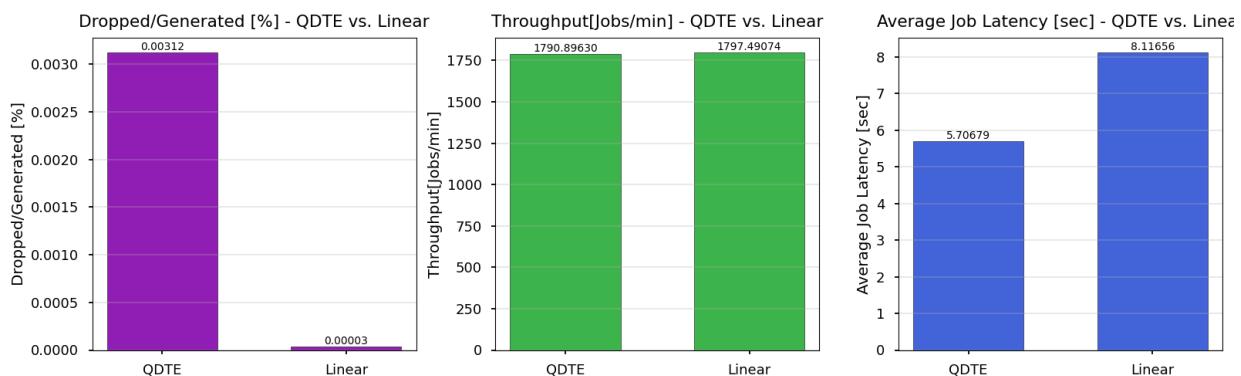
כעת נשווה בין הkonfiguraciot הטובות ביותר של הפונקציה הליינארית (Linear ramp-up, averaged decay) של הפונקציה לשיעורן ההשניה (Queue drain-rate estimation).

נירץ את שניהם על 10 סידים, נמצע את התוצאות ונשווה בין הערכים שמתקבלים:



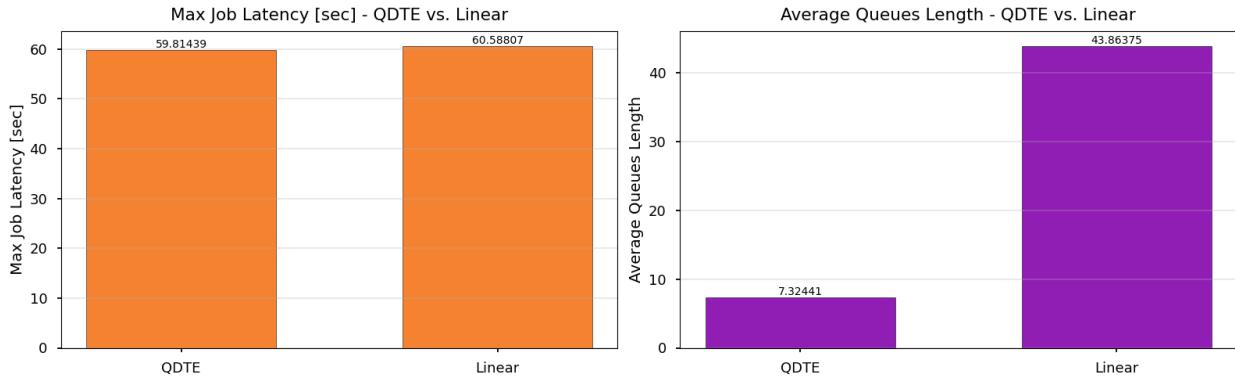
איור 7.31 : השוואת בין פונק' בקרה לינארית לשיעורן ההשניה, מדדי כמות רכיבים והודעות בקרה.

כאשר מסתכלים על כמות הרכיבים הממוצעת, QDTE משתמש ב-10% יותר רכיבים מאשר פונקציית הבקרה הלינארית. במקביל, כמות ה הודעות הבקרה שמנגנון זה שולח הם פי 4.6 מהמנגנון הלינארי. בשלוש המטריקות הללו, פונקציית הבקרה הלינארית טובת יותר – בכמות הרכיבים באופן שולי, ובכמות ההודעות בצורה דרסטית.



איור 7.32 : השוואת בין פונק' בקרה לינארית לשיעורן ההשניה, מדדי שיעור איבוד חבילות, תפוקה לדקה והשניה ממוצעת.

גם במדד איבוד ההודעות לפונקציה הליינארית יש יתרון, אם כי שיעור האיבוד במנגנון-QDTE הינו טוב מאוד גם כן – 0.3% מההודעות. לעומת זאת, **ההשניה הממוצעת עבור QDTE משמעותית יותר נמוכה מאשר המנגנון הלינארי (ירידה של 30%)**, ואילו התפוקה של שני המנגנונים דומה למדוי.



אייר 7.33 : השוואה בין פונק' בקרה לינארית לשיעורן השהיה, מדדי השהיה מקסימלית ואורך תור ממוצע כמו כן, ההשיה המקסימלית בין שני המנגנונים דומה למדיי, עם יתרון קל של ~שנייה לטובת מנגנון ה-QDTE. מבחינת אורך התורים הממוצע, כאן יש יתרון מובהק למנגנון ה-QDTE, שמשיג בממוצע אורך תור נמוך פי 6 (!) מהמנגנון הלינארי. מדד זה מייצג כי המנגנון מתמודד בצורה טוביה יותר עם העומסים, ולא יוצר מצבים של הצברותות גדולה מדי של הודיעות בתורים, מה שיכל לאורך זמן לגורום להשיהות גבוות יותר.

8. סיכום ומסקנות

במהלך העבודה על פרויקט זה, למדנו כיצד לעבוד עם `+TNeM0` כדי ליצור סימולציות מורכבות, להוציא מגוון סטטיסטיות ולעבד אותן כדי לקבל תובנות על המערכת אותה אנחנו מסמלרים, שבדרך כלל יהיה קשה ותועני יותר להפיך. בנוסף, היות והקוד לשימולטור צריך להיות ב-`C++`, היה צריך ללמוד את השפה הניל בצורה מסוימת בעבר עם הסימולטור,ידע שלחלקו לא היה לפני שהתחלו את הפרויקט. הצלחנו למדוד את המערכת האמיתית עליה התבססו מתוך סביבת הסימולטור בהצלחה, כפי שניתנו לראות מבדיקה השפויות שנעשה (התאמה בין תוצאות הסימולציה לתוצאות התיאורטיות) ולשגור התקלה באופן זהה לאופן בה היא מתרכשת במערכת האמיתית.

הצלחנו גם ליצור בסימולציה סטטיסטית מצב יציב של המערכת, בرمות עומס שונות, תוך כדי השגת זמן השהייה נמוך ואפס איבודי עבוזות, כרצוי במערכת האמיתית. לאחר מכן, כתבנו מנגנון לSIMULACRA DINAMICA בו כמות הרכיבים הפעילים משתנה עפ"י פונקציית בקרה בשליטות המלה ובחנו 2 גישות (פונקציות) שונות להשגת יודי ביצועים שהגדנו בעורת פונקציית ציון המתיחסת לביצועים במדדים השונים עם סדר עדיפות ביןיהם (כמפורטל ציון בכל אחד מן המדדים).

ראינו כי מנגנון ה-QDTE הינו אגרסיבי יותר, במיוחד בكونפיגורציה שנבחרה, בה הוא שואף להשיג השהיות נמוכות מאוד (הקרובות להשייה שהגענו בסימולציה הסטטיסטית). אגרסיביות זו בא לידי ביטוי בכמות הודעות בקרה גבוהה וביתר רכיבים במוצע, אך גם משיגה את יעד – ההשייה המומוצעת ואורך התורים נמוך משמעותית מאשר בפונקציה הלינארית. מנגד, המנגנון הלינארי מושג במוצע פחות רכיבים ומגיע לתפקה גדולה כמעט עם שיעור איבוד הودעות אפסי.

אין מנצח ברור מבין שתי הקונפיגורציות הניל, כאשר כל אחת נותנת מענה טוב יותר לממדדים אחרים וכושלת בממדדים בהם השניה מובילה.

אם אנחנו מעדיף מתעדפים השהייה נמוכה – נעדיף את פונקציית ה-QDTE, ואילו אם אנחנו מעדיף פרחות יחידות פעילות בממוצע, כמו עלות כלכלית נמוכה יותר – הפונקציה הלינארית עדיפה.

לא משנה באילו מבין שתי הפונקציות נבחר, יש עדיפות ברורה לגישה הדינמית על פני שימוש בכמות קבועה (סטטיסטית) של רכיבים, כפי שעשינו בסימולציה הסטטיסטית – שם השתמשנו ב-188 רכיבים (מתאימים + קטלוגים).

לשימוש בכל רכיב יש עלות גובהה, בתרחיש המציאות ממנו שאבנו השראה – **\$337 לחודש**.

- באמצעות פונק' הAKERה הלינארית השתמשנו (בממוצע) ב-142 = $[133.90944 + 7.40766]$ רכיבים – **חיסכון של 46 רכיבים, שהם \$15,502 בחודש**.

- באמצעות פונק' הAKER QDTE השתמשנו (בממוצע) ב-156 = $[145.92693 + 9.99883]$ רכיבים – **חיסכון של 32 רכיבים, שהם \$10,784 בחודש**.

לסיכום, מנגנון בקרה דינמי עדיף על פני קביעה סטטיסטית, ואילו מבין המנגנונים שבדקנו (או מנגנונים אחרים) עדיף – תלוי בסדר העדיפויות והממדדים המעניינים את המשמש. בבעיה המקורית, כיוון שההוצאות הכלכלית היא המנייע המובייל והחשוב ביותר, הינו ממליצים להשתמש במנגנון הAKER הלינארי בكونפיגורציה האופטימלית שמצאנו.