리눅스 커널 파라미터 튜닝을 통한 네트워크 성능 변화 분석

정규병 [1], 양경식 [2]
[1] 고려대학교 빅데이터융합학과 석사과정
[2] 고려대학교 컴퓨터학과 조교수

rbquddl16@korea.ac.kr, g_yang@korea.ac.kr

Network Performance Analysis through Linux Kernel Parameter Tuning

Kyu Byeong Jeong[1], Gyeongsik Yang[2]
[1]Dept. of Big Data Convergence, Korea University
[2]Dept. of Computer Science and Engineering, Korea University

요 약

본 연구는 리눅스 커널 파라미터 튜닝이 네트워크 성능 향상에 미치는 효과를 검증하는 것을 목표로 한다. 최근 이더넷 기술의 발전과 컴퓨팅 자원의 증가에도 불구하고, 성능 저하 문제는 여전히 존재하며, 그 주요 원인으로 리눅스 네트워크 스택 내부의 패킷 처리 지연과 자원 병목이 지적된다. 이에 따라 본 연구에서는 서비스 유형을 지연 민감 서비스와 처리량 민감 서비스 두 가지로 분류하고, 각 유형에 적합한 커널 파라미터를 도출 및 튜닝하여 성능 변화를 실험적으로 분석하였다. 실험 결과, 지연 감소와 처리량 향상 등에서 유의미한 개선이 확인되었으며, 이는 커널 수준 튜닝이 서비스 특성에 따른 네트워크 성능 최적화에 효과적으로 기여할 수 있음을 시사한다.

1. 서론

최근 실시간 웹 서비스, 클라우드 백업, AI 모델 학습 등 서로 다른 트래픽 특성을 가지는 AI 기반 플랫폼의 확산으로 IT 서비스 전반에서 대용량 데이터 처리와 지연 최소화 등 보다 정교한 네트워크 성능 요건이 요구되고 있다[1]. 이에 따라 대역폭과 전송 속도가 지속적으로 개선되고 있다.[2] 그럼에도 불구하고 운영체제 내부의 네트워크 스택에서 발생하는 처리 지연과 자원 병목은 여전히 전체 서비스 성능 저하의 주요 원인으로 지적된다.[3][4]

대표적인 오픈소스 운영체제인 리눅스는 커널 수준에서 다양한 네트워크 제어 파라미터를 제공한다. 시스템 운영자는 이를 튜닝하여 소켓 처리, 큐 관리, 인터럽트 분산 등 네트워크 처리 임계치와 단위를 최적화할 수 있다. 그러나 실제 운영 환경에서는 커널 파라미터에 대한 이해 부족으로 기본 설정을 그대로 사용하는 경우가 일반적이며[5], 이는 다양한 워크로드를 효과적으로 처리하지 못하거나 성능 저하로 이어질 수 있다[6][7]. 이러한 한계는 워크로드 특성에 따른 커널 파라미터 최적화 전략의 필요성을 시사한다.

이에 본 연구는 다양한 네트워크 기반 응용 서비스를 성능 요구에 따라 분류하고, 각 유형별로 어떤 파라미터 튜닝이 유의미한 성능 개선으로 이어지는지 분석한다. 또한 파라미터 간 상호작용과 Trade-off 를 고찰함으로써, 파라미터 기반 최적화의 적절성과 실효성을 평가한다.

2. 본론

2.1. 서비스 유형 분류 및 트래픽 특성 정의

본 연구에서는 지연 민감형과 처리량 민감형 두 가지 서비스로 분류하였다. 지연 민감형 서비스는 개별 요청에 대해 수 밀리초 수준의 응답 지연을 허용 한계로 가지며, 대표적으로 실시간 웹 서비스와 결제 시스템이 이에 해당한다. 이러한 서비스는 짧고 빈번한 트랜잭션을 수반하며, 특히 상위 0.1% 요청 지연이 사용자 경험에 직접적인 영향을 미친다.[8][9] 반면 처리량 민감형 서비스는 단일 요청의 지연보다는 전체 처리 속도를 중시하며, 대표적으로 백업, 로그 수

집 등이 있다. 이들 워크로드는 긴 세션, 큰 전송 단위, 지속적인 자원 점유가 특징이다.[10][11]

이에 따라 두 유형은 성능 목표와 트래픽 특성이 뚜렷이 구분되며, 커널 파라미터의 튜닝 방향과 성능 영향 또한 달라진다. 표 1 은 트래픽 특성, 핵심 성능 지표, 대표 서비스를 중심으로 비교한 것이다.

<표 1> 서비스 유형 분류

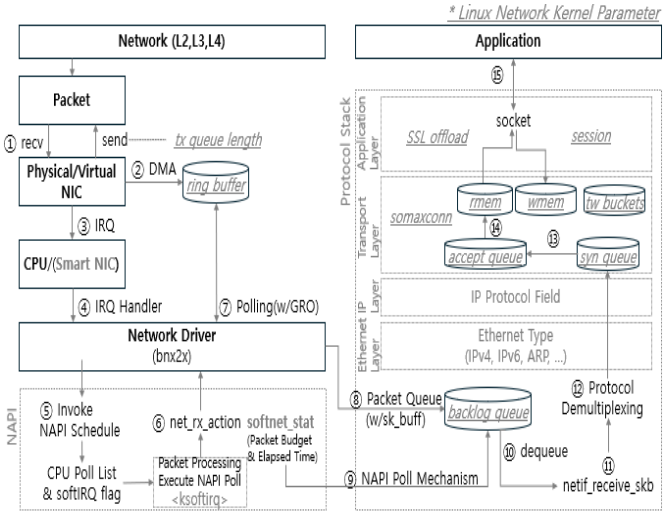
유형	지연 민감	처리량 민감
트래픽 특성	1) 빠른 응답 2) 짧은 세션 3) 동시 접속성	1) 대용량 처리 2) 장시간 연결 3) I/O 최적화/ 안정성
성능 지표	1) 응답시간 2) 지연시간 3) 재전송 수	1) 처리량 2) Jitter 3) 패킷 손실률
대표 서비스	1) 실시간 웹 2) 결제 시스템 3) 온라인 게임	1) 백업/ 대용량 전송 2) AI 모델 학습 3) VOD 스트리밍

2.2. 서비스 유형별 파라미터 선정 및 평가방법

먼저 네트워크 성능에 직접적인 영향을 미치는 약 20 여 종의 커널 파라미터를 후보군으로 선정하였다. 선정 기준은 리눅스 네트워크 스택의 핵심 기능 영역(TCP 및 UDP 통신 처리, 큐 관리와 버퍼링, 세션 유지 및 혼잡 제어 등)에 대한 이론적 분석과, 기존 문헌에서 성능과의 상관성이 강조된 항목을 우선적으로 고려하였다.[12][13] 이후 실험을 통해 각 파라미터 별 성능 영향도를 평가하였다.

지연 민감형 서비스의 경우 응답 지연 최소화가 핵심이므로 연결 대기 큐, 세션 설정 및 종료와 관련된 파라미터를 중심으로 성능 개선 효과를 검증하였다.

(그림 1) Linux Network Stack 및 Parameter 정리



반면 처리량 민감형 서비스는 높은 처리량과 전송 효율성이 중요하므로 네트워크 인터페이스 큐의 처리 성능, 송수신 버퍼 크기와 같은 전송 효율 관련 파라미터들로 검증하였다.(그림 1) 그 결과 각 서비스 유형에 적합한 핵심 파라미터를 도출하였으며 성능 개선 목적, 주요 지표, Trade-off 항목과 함께 표 2 와 표 3 에 정리하였다.

<표 2> 지연 민감 서비스의 파라미터

파라미터	최적화 목적	성능 지표	Trade-off
somaxconn.syn_backlog, max_syn_backlog	TCP 연결대기 큐 증가, SYN 큐 확장	Packet 수신/ 전송 처리 성능	메모리 사용률↑, 지연 증가 가능성↑
fin_timeout, tw_reuse, tw_buckets	세션 종료 및 Time Wait 관리	동시접속 세션 처리	안정성 저하
net.ipv4.tcp_low_latency	지연 최소화	지연	처리량 감소 가능성↑

<표 3> 처리량 민감 서비스의 파라미터

파라미터	최적화 목적	성능 지표	Trade-off
net.core.rmem, net.core.wmem, max_backlog	송수신 버퍼 확장, 네트워크 큐 확장	처리량, 패킷수신률	메모리 사용률↑
net.core.netdev_budget	NAPI 처리 패킷 수 증가	처리량	지연 증가 가능성↑
Ring Buffer, txqueuelen	NIC 송수신 큐 확장	대역폭	메모리 사용률↑

3. 실험

3.1. 실험 구성

파라미터에 따른 성능 최적화를 분석하기 위한 구체적인 환경은 표 4 에 제시하였다. 특히 클라이언트에서의 자원 부족 등 병목이 발생하지 않도록 구성하여, 서버 쪽 커널 파라미터 튜닝 자체에서 기인하는 효과를 명확히 관찰할 수 있도록 하였다.

<표 4> 실험 환경 구성

구분	내용
서버 스펙	24 Core 64 GB Memory
OS	Ubuntu 22.04.5
Kernel	5.15.0-151-generic
CPU Model	Intel Xeon E5-2650 v4
NIC Driver	igb 1.61
웹 서비스	nginx 1.18.0
클라이언트 병목 제거	-ulimit -n 1048576 적용 -conntrack 비활성화
모니터링 명령어	-netstat -s | egrep -i 'retran|resent|segments retransmitted' -i -ethtool -S eno1 | egrep 'err|drop|fifo|crc|miss|timeout' -i -dmesg | egrep -i 'eno1|eth|net|ixgbe|mlx|e1000' -grep . /proc/net/softnet_stat

성능 측정은 지연 민감형 서비스와 처리량 민감형 서비스 두 가지 유형을 대상으로 수행하였다.

첫째, 지연 민감형 서비스의 특성을 고려하여 실시간 웹 서비스 환경과 유사한 조건에서 성능을 측정하였다. 이를 위해 iperf3 를 사용하여 128 개의 병렬 스트림과 16KB 전송 단위를 설정하고, 재전송 발생 여부를 확인하였다. 또한 HTTP 기반 요청 처리 성능은 ab 와 wrk 를 활용하였다. ab 는 50,000 건의 요청을 5,000 개의 동시 접속으로 전송하여 응답 지연과 평균 응답 시간을 측정하였으며, wrk 는 12 개의 스레드와 500 개의 세션을 30 초 동안 유지하여 응답 지연 및 처리 성능을 평가하였다.

둘째, 처리량 민감형 서비스의 특성을 반영하여 대규모 데이터 전송 성능을 검증하였다. 이를 위해 iperf3 의 UDP 모드를 사용하였으며, 전송률 600Mbps, 단위 크기 1200 바이트, 병렬 스트림 4 개 조건으로 테스트를 수행하여 전송량, Jitter, 패킷 손실률을 측정하였다. 또한 HTTP 기반 대량 요청 처리 성능은 ab 와 wrk 를 통해 평가하였다. ab 는 50,000 건의 요청을 5,000 개의 동시 접속으로 전송하여 전송량과 처리율을 측정하였으며, wrk 는 12 개의 스레드와 500 개의 세션을 60 초간 유지하며 전송량 및 처리율을 수집하였다.

ab 와 wrk 는 두 유형의 서비스 모두에서 동일한 옵션으로 테스트를 수행하였으며, 이를 통해 커널 파라미터 튜닝이 지연 민감 서비스와 처리량 민감 서비스 사이에서 어떤 상충 관계를 만들어내는지를 분석하였다. 즉, 특정 파라미터 설정이 지연 시간 개선에는 유리하지만 처리량 감소를 초래하거나, 반대로 처리량 향상에는 기여하나 지연을 악화시키는 경향을 정량적으로 확인하였다. 아울러 본 연구에서 사용된 성능 측정 도구와 세부 명령어는 표 5 에 정리하여 제시하였다.

<표 5> 성능 측정 도구 및 명령어

유형	측정도구	주요 옵션 및 명령어
지연 민감	iperf3(TCP)	iperf3 -c [server ip] -p 5201 -t 60 -P 128 -l 16K
처리량 민감	iperf3(UDP)	iperf3 -c [server ip] -p 5201 -t 60 -P 4 -u -b 600M -l 1200
공통	Apache Bench	ab -n 50000 -c 5000 http://[server ip]/index.html
	wrk	wrk -t12 -c500 -d30s http://[server ip]

3.2. 실험 결과

표 6 는 각 서비스 유형에 적용한 커널 파라미터의 설정 값을 정리한 것으로, 본 연구에서는 여러 파라미터를 동시에 조합하여 변경하면서 성능을 측정하고, 그 결과를 토대로 최적의 설정 값을 선별하였다. 특히 성능 측정 도구를 통해 직접 관측 가능한 지표를 기준으로 파라미터를 선별하였다. 예를 들어, TCP keepalive 와 관련된 파라미터는 동시 접속 환경에서 효과를 발휘하나, 해당 효과를 정량적으로 관측하기 어려워 실험 대상에서 제외하였다.

<표 6> 커널 파라미터 설정 값

유형	커널 파라미터	기본 값	튜닝 값
지연 민감	net.core.somaxconn	4096	8192
	net.ipv4.tcp_max_syn_backlog	4096	8192
	net.ipv4.tcp_fin_timeout	60	15
	net.ipv4.tcp_tw_reuse	2	1
	net.ipv4.tcp_max_tw_buckets	262144	1048576
	net.ipv4.tcp_low_latency	0	1
처리량 민감	net.core.rmem_default	212992	67108864
	net.core.rmem_max	212992	67108864
	net.core.wmem_default	212992	67108864
	net.core.wmem_max	212992	67108864
	net.core.netdev_max_backlog	1000	10000
	net.core.netdev_budget	300	600
	ethtool -G [NIC] rx, tx	256	4096
	txqueuelen	1000	5000

지연 민감 서비스의 실험 결과, 표 7 와 같이 커널 파라미터 튜닝 이후 실패 요청 수가 약 15% 감소하였으며, 최대 지연의 약 47% 개선되는 등 안정성이 향상되었다. 반면, 평균 응답시간은 baseline 대비 0.9% 소폭 증가하였는데, 이는 통계적으로 유의미한 수준은 아니며, 튜닝 과정에서의 커널 파라미터 조정으로 인해 재전송 억제 및 실패 요청 감소 과정에서 일부 응답 구간이 길어지는 trade-off 가 발생했을 가능성이 있다. 그러나 p99 구간의 응답 지연은 효과적으로 억제됨을 확인할 수 있었으며, 이는 전체적인 서비스 품질 관점에서 성능의 저하보다는 안정성 개선이 더 두드러진 결과로 해석된다.

<표 7> 지연 민감 튜닝 효과 요약

측정항목	튜닝 전 (평균±표준편차)	튜닝 후 (평균±표준편차)	변화율
평균 응답시간(ms)	170.7±4.0	172.3±1.8	0.9% 증가
최대 응답시간(ms)	227.6±13.5	224.8±7.4	1.2% 감소
평균 지연(ms)	7.50±3.86	7.50±2.35	6% 감소
최대 지연(ms)	최대 415	최대 218	47% 감소
실패 요청 수	55.5K±2.2K	46.7K±3.0K	16% 감소
재전송 수	66K	61K	8% 감소

한편, 처리량 민감 서비스에서는 표 8 와 같이 HTTP 처리량이 약 5% 증가하고, RPS 가 220K 수준에

서 231K 로 향상되었으며, UDP 손실률은 0%로 완전히 제거, jitter 는 0.002~0.008ms 수준으로 안정화되는 등 대규모 전송 환경에서 전송 효율성과 품질이 크게 개선되었다. 다만, 동시성 부하(ab, -c 5000)에서는 표 9와 같이 평균 및 최대 응답시간이 악화되는 Trade-off가 관찰되었다.

<표 8> 처리량 민감 튜닝 효과 요약

측정항목	튜닝 전	튜닝 후	변화율
처리량(MB/s)	75	79	5% 증가
RPS	220K	231K	5% 증가
Jitter(ms)	0.005~0.009	0.002~0.008	28.6% 감소
손실률(%)	0.025~0.051	0	100% 감소

<표 9> Trade-off 요약

지표 (도구)	지연 민감 튜닝	처리량 민감 튜닝
평균/최대 응답시간 (ab)	Baseline 과 동일	10~15% 응답시간 증가

종합적으로, 지연 민감 튜닝은 응답시간 분포의 안정화와 실패율 감소에 효과적이었고, 처리량 민감 튜닝은 처리량 및 전송 신뢰성 최적화에 기여하였다. 이는 커널 파라미터 튜닝이 서비스 유형별 성능 특성에 따라 상이한 영향을 미치며, 결과적으로 워크로드 성격에 따라 서로 다른 최적화 전략이 필요함을 실증적으로 보여준다.

4. 결론

본 연구는 리눅스 커널 파라미터가 서비스의 트래픽 특성에 따라 네트워크 성능에 미치는 영향을 실험적으로 규명하였으며, 서비스 유형별로 주요 파라미터를 선별 및 튜닝함으로써 실질적인 성능 개선 가능성을 확인하였다. 이를 통해 네트워크 성능 최적화 과정에서 커널 파라미터의 전략적 중요성을 정립하였다는 데 의의가 있다.

다만, 본 연구는 정적 환경과 제한된 서비스 유형에 초점을 맞추었기에, 보다 다양한 워크로드와 복합적인 운영 환경에 적용하기에는 제약이 존재한다. 향후 연구에서는 트래픽 특성의 실시간 분석을 기반으로 한 자동화된 동적 튜닝 기법, 머신러닝 기반 파라미터 최적화 및 추천 모델, 그리고 대규모 분산 환경에서의 검증을 통해 본 연구의 실용성과 일반화를 한층 강화할 수 있을 것이다. 이를 통해 커널 수준의 네트워크 최적화가 이론적 연구를 넘어 실제 서비스 환경에서 적용 가능한 지능형 성능 관리 체계로 발전할 수 있을 것으로 기대된다.

Acknowledgement

이 논문은 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원(RS-2024-00336564)과 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원(RS-2024-00405128)을 받아 수행된 연구임.

참고문헌

[1] Yoo, Yeonho, et al. "Revisiting Traffic Splitting for Software Switch in Datacenter." Proceedings of the ACM on Measurement and Analysis of Computing Systems 9.2 (2025): 1-26.

[2] IEEE 802.3bs Task Force, "IEEE Standard for Ethernet Amendment 5: Media Access Control Parameters, Physical Layers, and Management Parameters for 200 Gb/s and 400 Gb/s Operation," IEEE Std 802.3bs-2017, pp. 1–372, Dec. 2017.

[3] R. Agarwal, A. Narayanan, S. Katti, S. Shenker, and S. Ratnasamy, "Understanding Host Network Stack Overheads," ACM SIGCOMM Computer Communication Review, vol. 51, no. 3, pp. 50–57, 2021.

[4] Choi, Wonmi, et al. "Intelligent Packet Processing for Performant Containers in IoT." IEEE Internet of Things Journal (2024).

[5] Y. Cui, Q. Chen, and J. Yang, "EOS: Automatic In-vivo Evolution of Kernel Policies for Better Performance," arXiv preprint arXiv:1508.06356, 2015.

[6] M. Mathis, J. Heffner, and R. Reddy, "Web100: Extended TCP Instrumentation for Research, Education and Diagnosis," ACM Computer Communication Review, vol. 33, no. 3, pp. 69–79, 2003.

[7] Yoo, Yeonho, et al. "Machine learning-based prediction models for control traffic in SDN systems." IEEE Transactions on Services Computing 16.6 (2023): 4389-4403.

[8] T. Dunigan, M. Mathis, and B. Tierney, "A TCP Tuning Daemon," in Proc. of the ACM/IEEE Conference on Supercomputing (SC), vol. 4, pp. 1–16, Nov. 2002.

[9] J. Dean and L. A. Barroso, "The Tail at Scale," Communications of the ACM, vol. 56, no. 2, pp. 74–80, 2013.

[10] A. Shpiner, A. Valadarsky, M. Ben-Yehuda, A. Schuster, and R. Kat, "QJUMP: Effective Packet Scheduling for Datacenter Networks," in Proc. of the USENIX Symposium on Networked Systems Design and Implementation (NSDI), vol. 12, pp. 1–14, 2015.

[11] C. Delimitrou, D. Lo, C. Kozyrakis, et al., "Heracles: Improving Resource Efficiency at Scale," in Proc. of the IEEE/ACM Int. Symp. on Computer Architecture (ISCA), vol. 41, pp. 450–462, 2015.

[12] R. Radhakrishnan, L. Cheng, J. Chu, A. Jain, and B. Raghavan, "TCP Fast Open," in Proc. of the 7th ACM CoNEXT Conference, pp. 21–32, 2011.

[13] J. Gettys and K. Nichols, "Bufferbloat: Dark Buffers in the Internet," Communications of the ACM, vol. 55, no. 1, pp. 57–65, 2012.