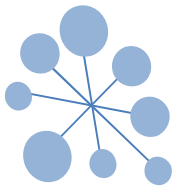


[奨励講演] vhost-userにおける キャッシュ作用の究明に向けた包括的評価

○竹谷 大地[†] 川島 龍太[†]
中山 裕貴^{††} 林 經正^{††} 松尾 啓志[†]

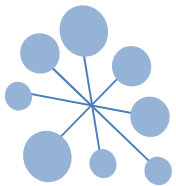
[†] 名古屋工業大学大学院

^{††} 株式会社ボスコ・テクノロジーズ

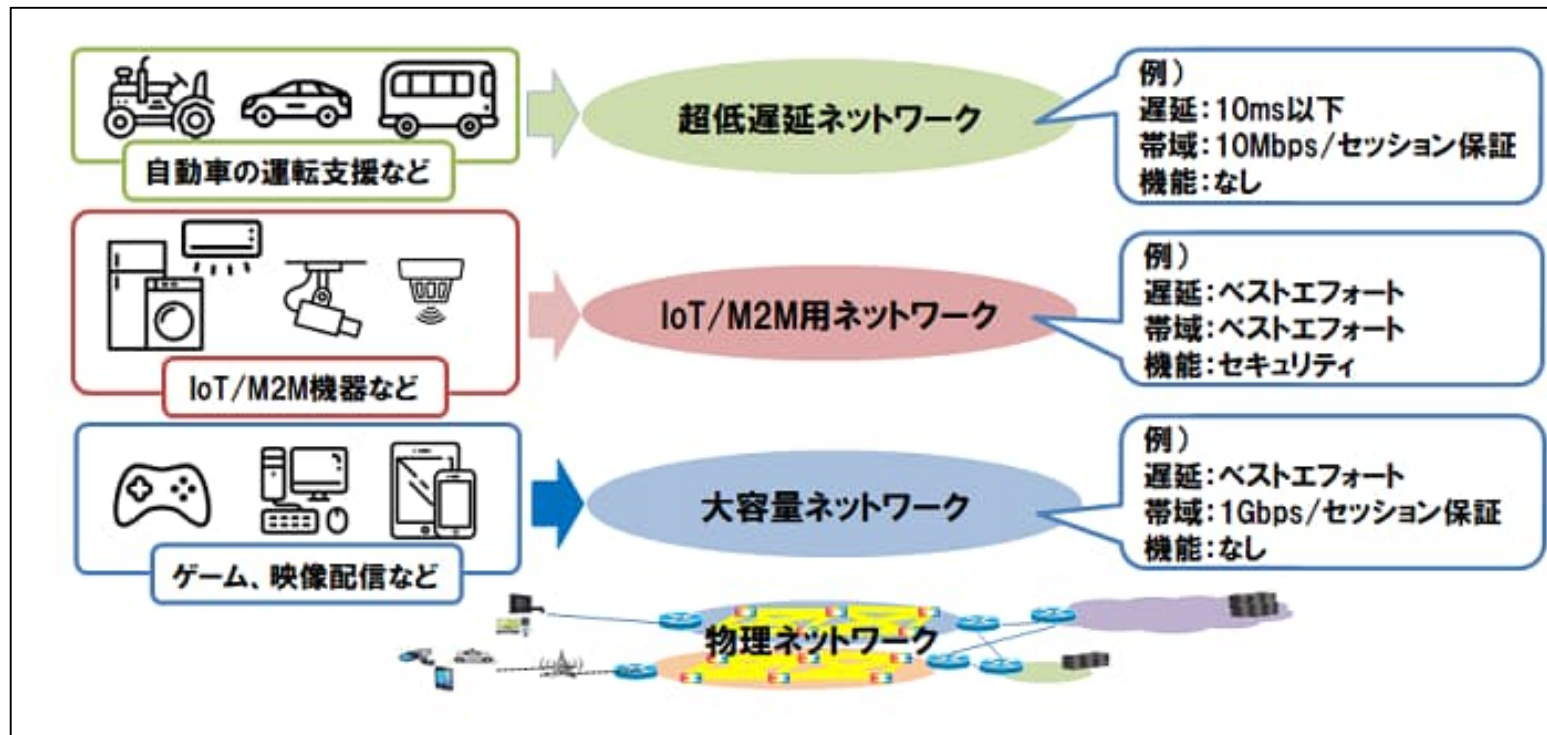


目次

- **研究背景**
- 関連研究
- 評価内容
 - 使用するプログラム
 - 評価項目
- **評価結果**
 - キャッシュ利用効率と性能の相関関係
 - 性能向上を実現した要因の特定
 - 仮想I/Oの更なる性能向上に必要な仕様の検討



クラウドネイティブなネットワーク



† NTT 研究開発「ネットワークスライシング技術～5G時代、お客様の要件を満たす新しいネットワークの構築を目指して～」

<https://www.rd.ntt/research/NW99-324.html>

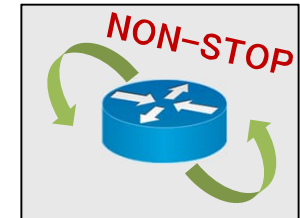
- ・ コアネットワークの運用を自動化
- ・ 迅速, 柔軟な構成変更

パケット処理性能が低い
(6Gにおける性能要求の10分の1)

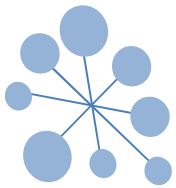
Cloud-Native Network Function



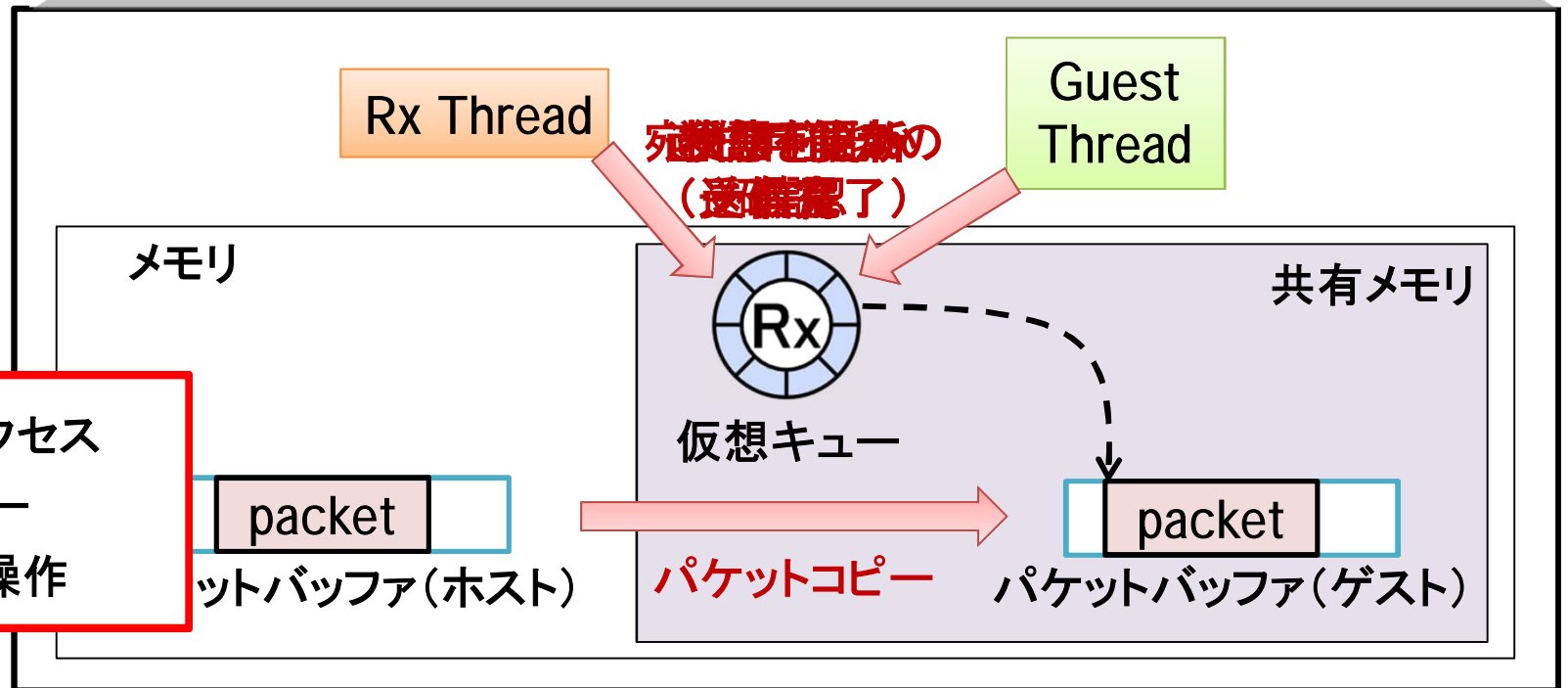
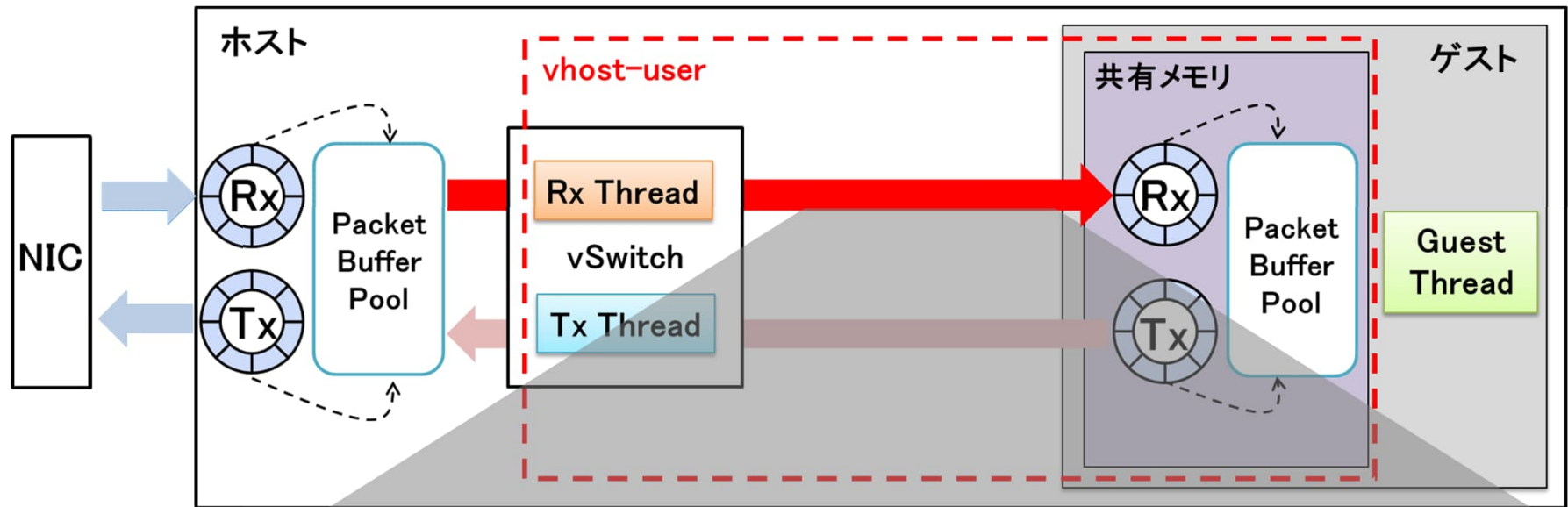
マルチテナント化が可能



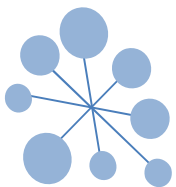
機能更新時も継続稼働



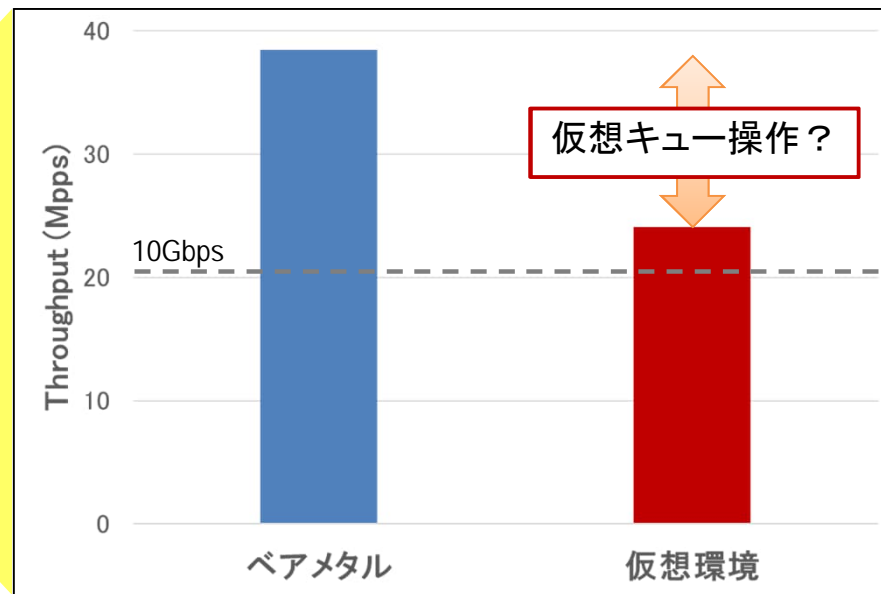
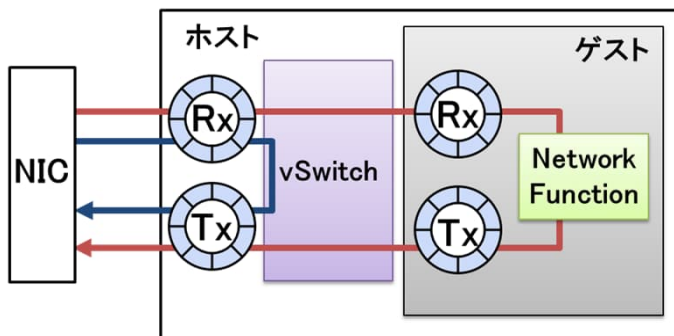
仮想環境におけるパケット処理



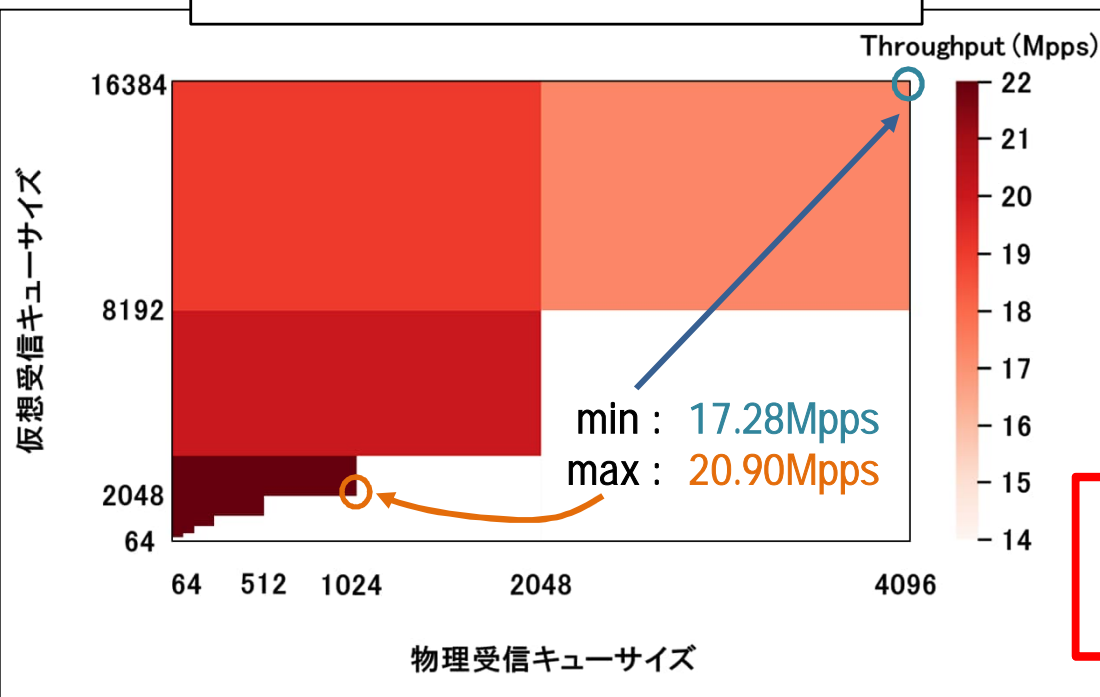
多数のメモリアクセス
・ パケットコピー
・ 仮想キュー操作



ベアメタルとのI/O性能差



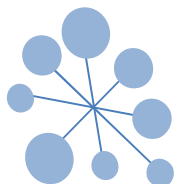
仮想キューサイズによる性能差



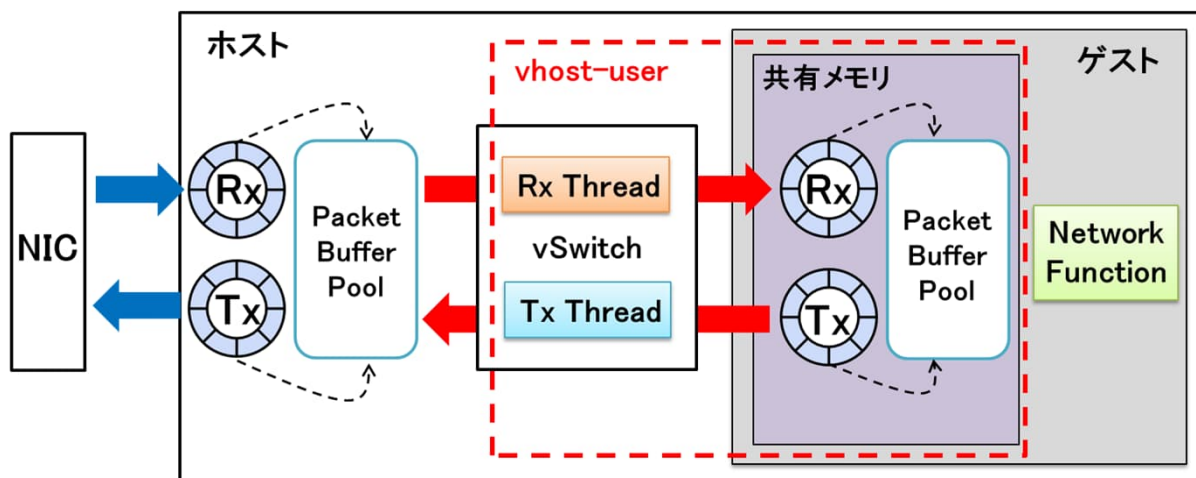
† R. Kawashima, "Software Physical/Virtual Rx Queue Mapping toward High-Performance Containerized Networking", IEEE Transactions on Network and Service Management, 2021

仮想受信キューサイズ	L1 Miss Latency
1024	5.1ns
8192	5.7ns

キャッシュ利用効率が
更なる性能向上の鍵？

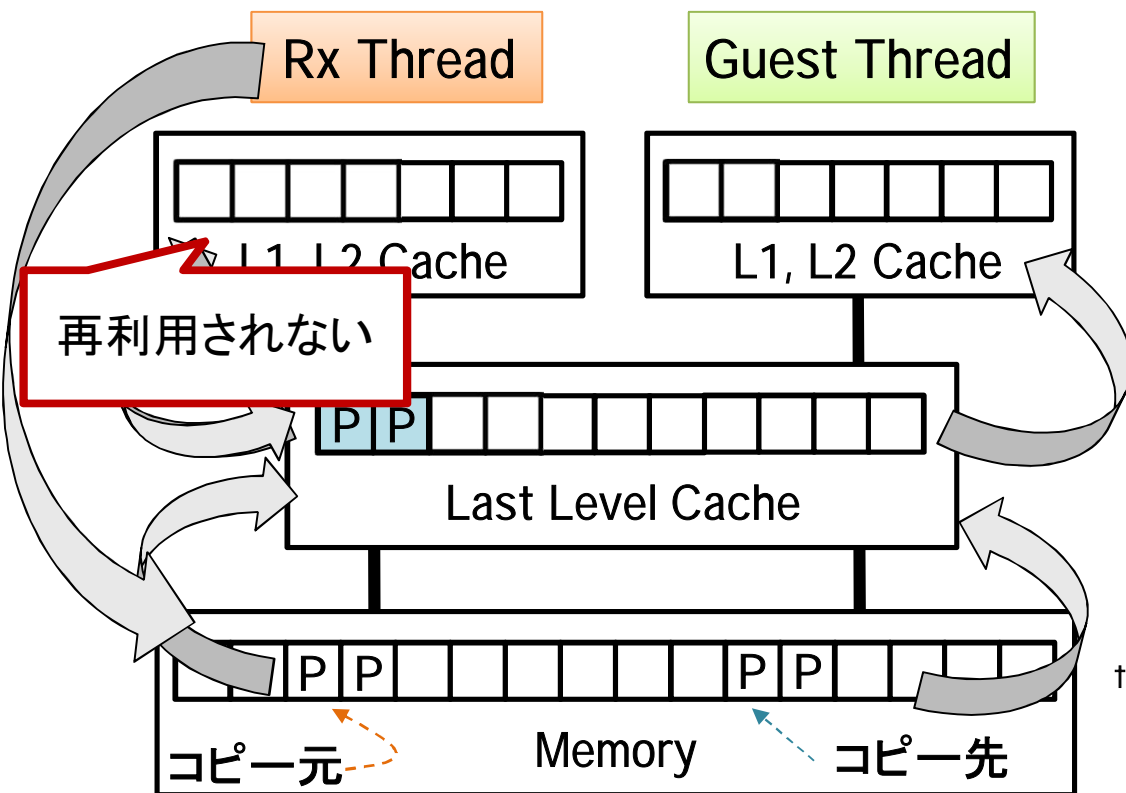


パケットコピーによるキャッシュ汚染解消



Non-temporal命令

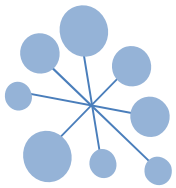
- キャッシュを利用しない load/store
- メモリに対して直接処理を実行



- キャッシュヒットレートの向上
- × スループットは変化せず

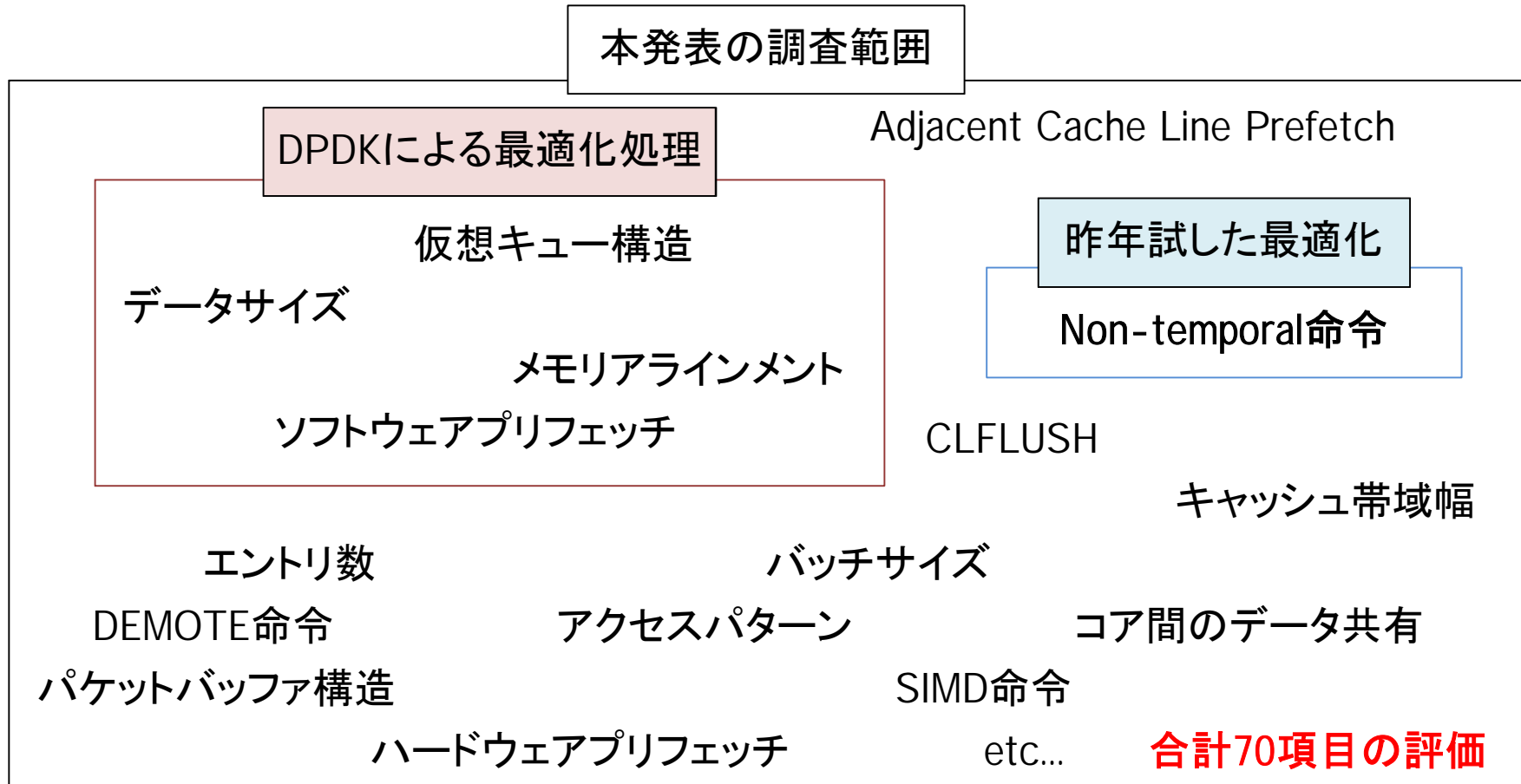
- Rxスレッドはボトルネックではなかった？
- メモリアクセス等の要因に相殺された？
- キャッシュ汚染の影響は小さい？

† 竹谷他, “仮想 I/O のキャッシュ汚染解消に向けた Non-Temporal命令の有効性評価”, 信学技報 (ICM2021-16)

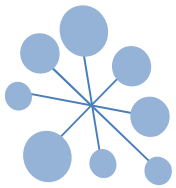


本発表の位置付け

- 仮想I/O性能とキャッシュ利用効率の関連性を解明

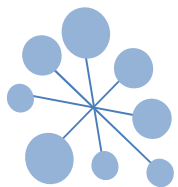


- ・ ベアメタルと同等の性能を実現可能か
- ・ 仮想I/Oのボトルネックはどこか



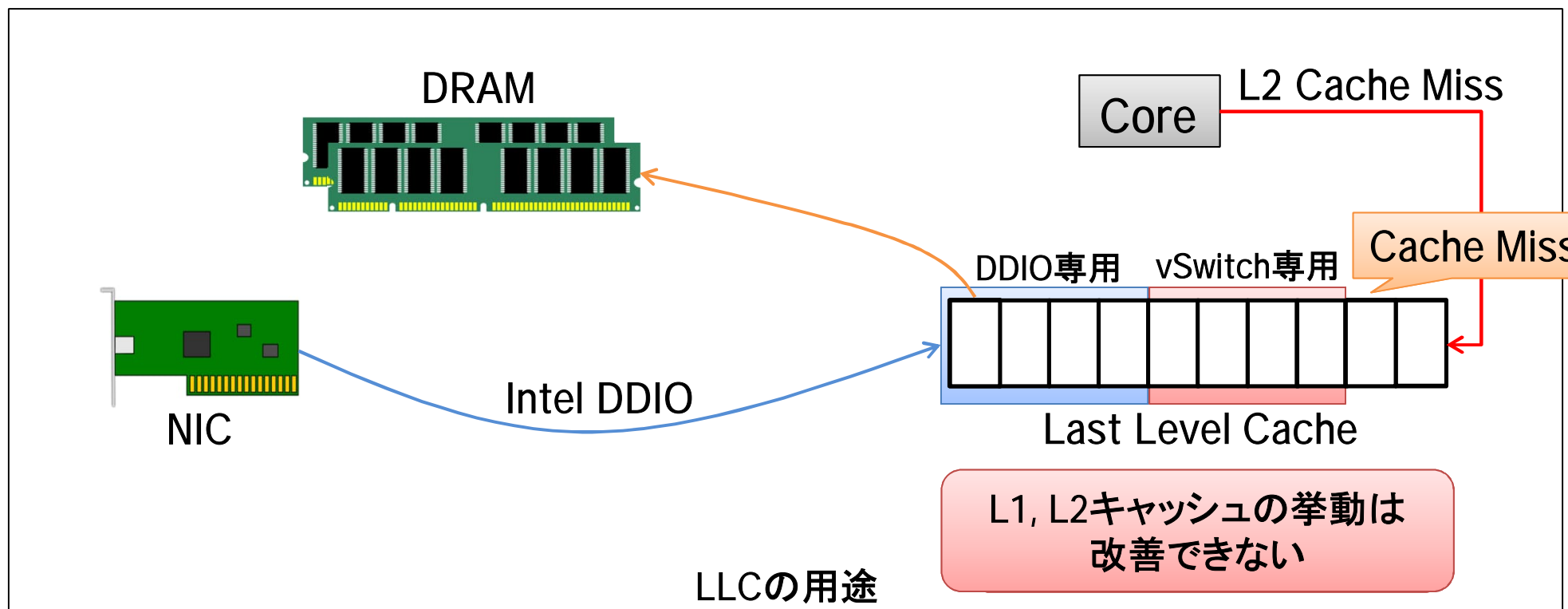
目次

- 研究背景
- 関連研究
- 評価内容
 - 使用するプログラム
 - 評価項目
- 評価結果
 - キャッシュ利用効率と性能の相関関係
 - 性能向上を実現した要因の特定
 - 仮想I/Oの更なる性能向上に必要な仕様の検討



ベアメタル環境におけるLLCの制御

- アプリケーションごとにLLCを排他的に割り当て
 - パケットと他コアのキャッシュデータとのLLC競合を防止

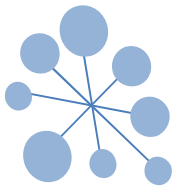


† A. Farshin, A. Roozbeh, G.Q.M. Jr., and D. Kostić,

“Reexamining direct cache access to optimize i/o intensive applications for multi-hundred-gigabit networks”, USENIX ATC 20

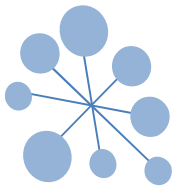
†† S. Thomas, R. McGuinness, G.M. Voelker, and G. Porter,

“Dark packets and the end of network scaling”, ANCS '18



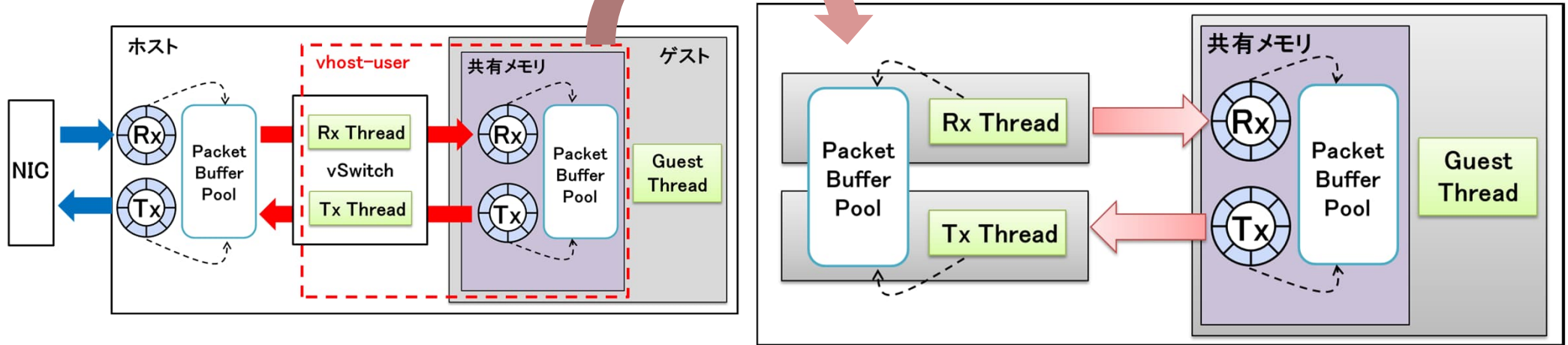
目次

- 研究背景
- 関連研究
- **評価内容**
 - 使用するプログラム
 - 評価項目
- 評価結果
 - キャッシュ利用効率と性能の相関関係
 - 性能向上を実現した要因の特定
 - 仮想I/Oの更なる性能向上に必要な仕様の検討

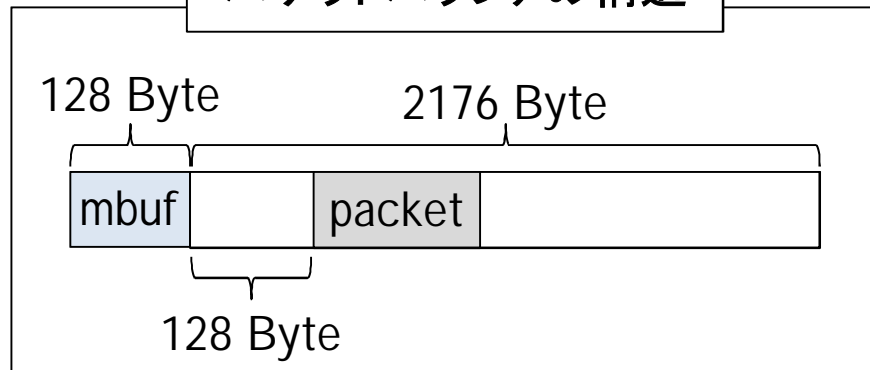


vhost-user (DPDK) のモデル化

本質的なパケット転送処理の影響を調査

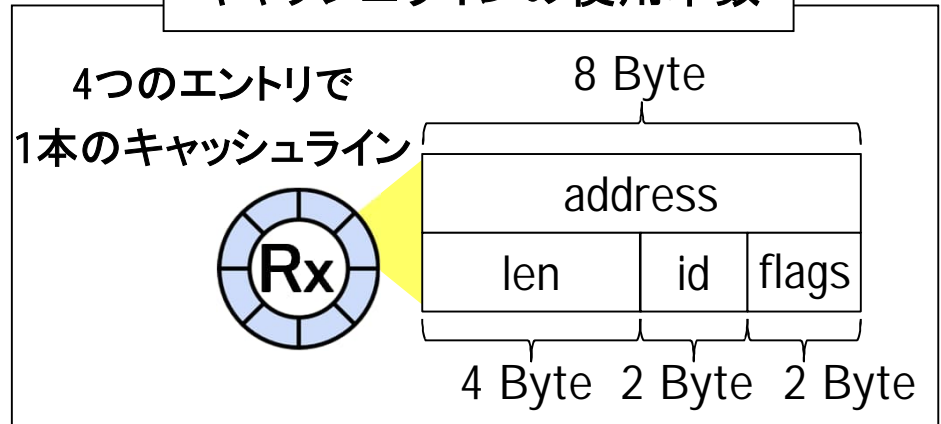


パケットバッファの構造

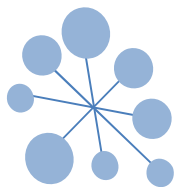


パケット処理性能(64Byte): 約14Mpps

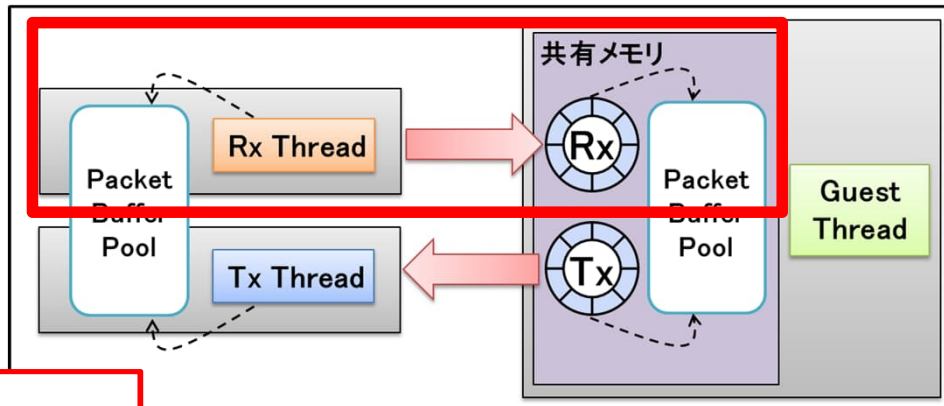
キャッシュラインの使用本数



DPDKを再現した実装



プログラムにおけるキャッシュの挙動

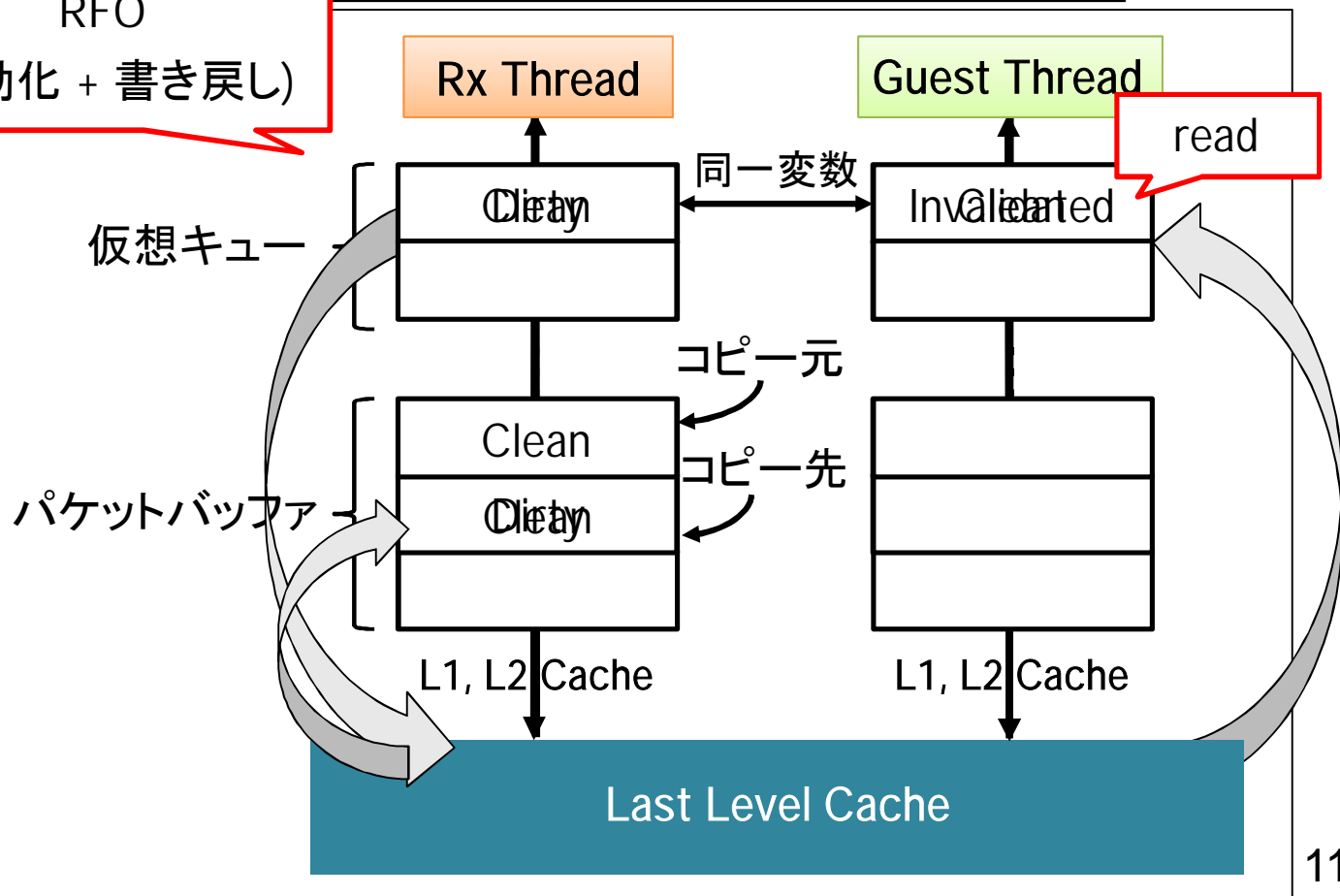


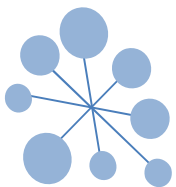
RFO
(無効化 + 書き戻し)

Dirty: LLCとの一貫性がない
Clean: LLCとの一貫性がある

送信フロー

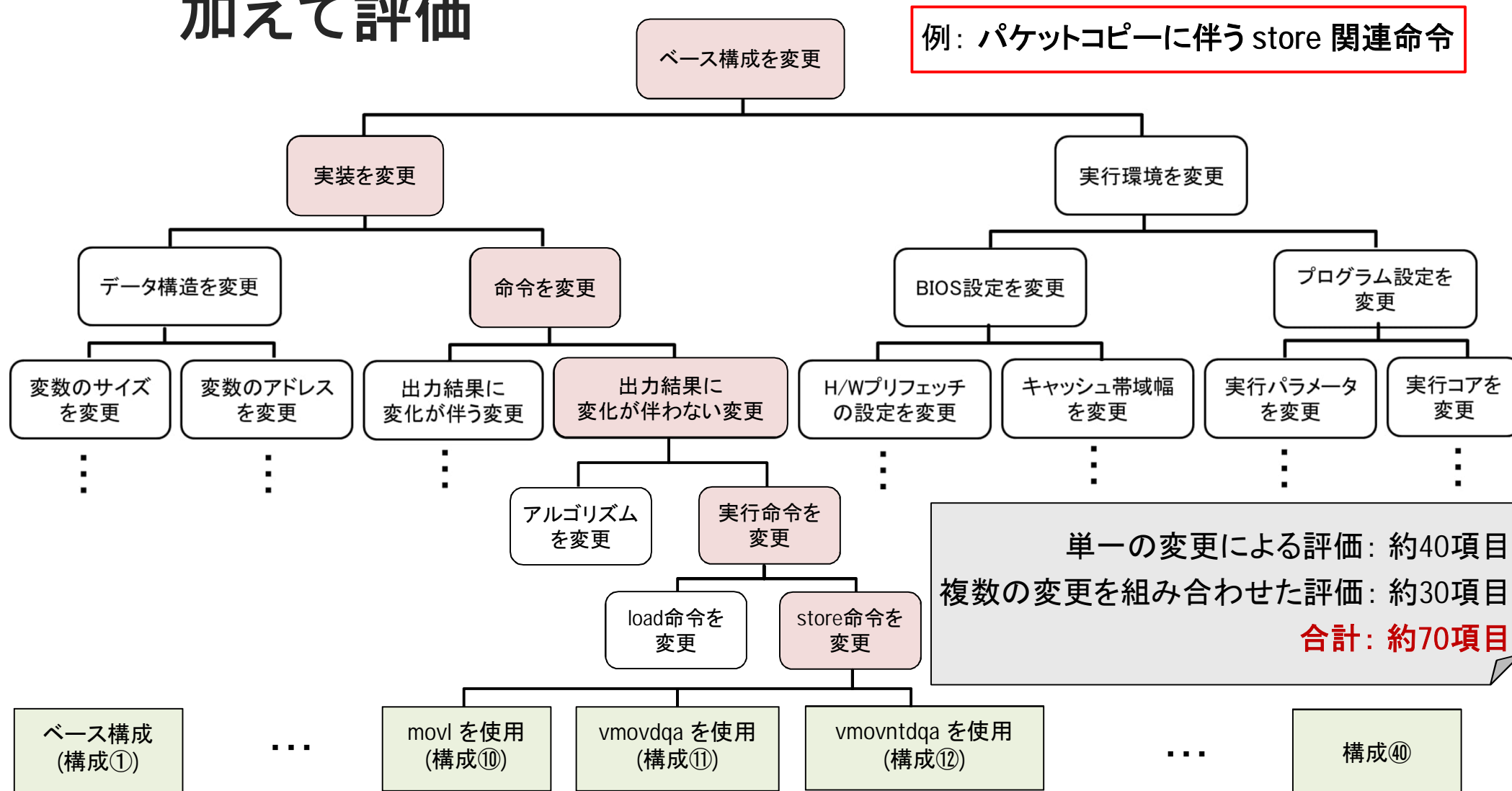
1. 送信可能か確認
2. パケットをコピー
3. 仮想キューを更新
4. 受信したアドレスを取得





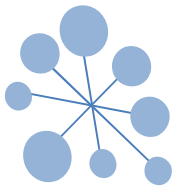
評価項目

- DPDKを模した「ベース構成」に様々な変化を加えて評価



DPDK相当

先行研究



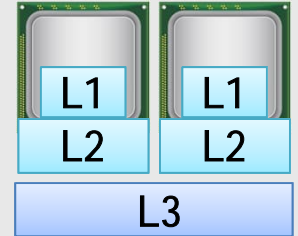
キャッシュ作用の解析手順

1. 各スレッドにおけるキャッシュと性能の相関を調査

Rx Thread

Guest Thread

Tx Thread



2. ベース構成との性能差を生じさせた要因を特定

Non-temporal命令？

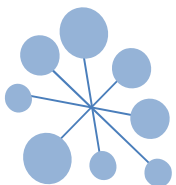
...

CLFLUSH？

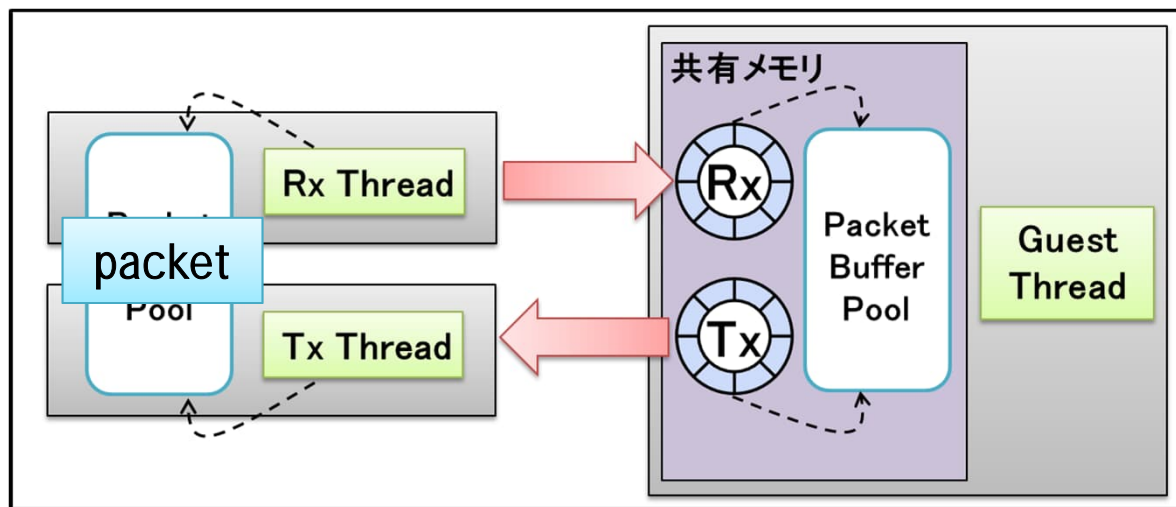
プリフェッチ？



3. 更なる性能向上の実現について検討



評価環境

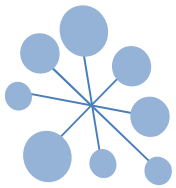


CPU	Intel Core i9-7940X CPU @ 3.10GHz 14 コア (HT 無効)
L1i Cache	32KB 8-way
L1d Cache	32KB 8-way
LFB Entries	10
L2 Cache	1024KB 16-way
Last level Cache	19712KB 11-way
memory	DDR4 2666MT/s
OS	AlmaLinux 8.5
perf	v4.18.0

測定イベントの一例

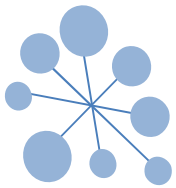
- ・ スループット
- ・ L1, L2, L3キャッシュへの総アクセス回数
- ・ L1, L2, L3キャッシュの総ミス数
- ・ プリフェッチ効率
- ・ LFBによるストールサイクル数
- ・ RFO(Read For Ownership) etc...

合計約20種類

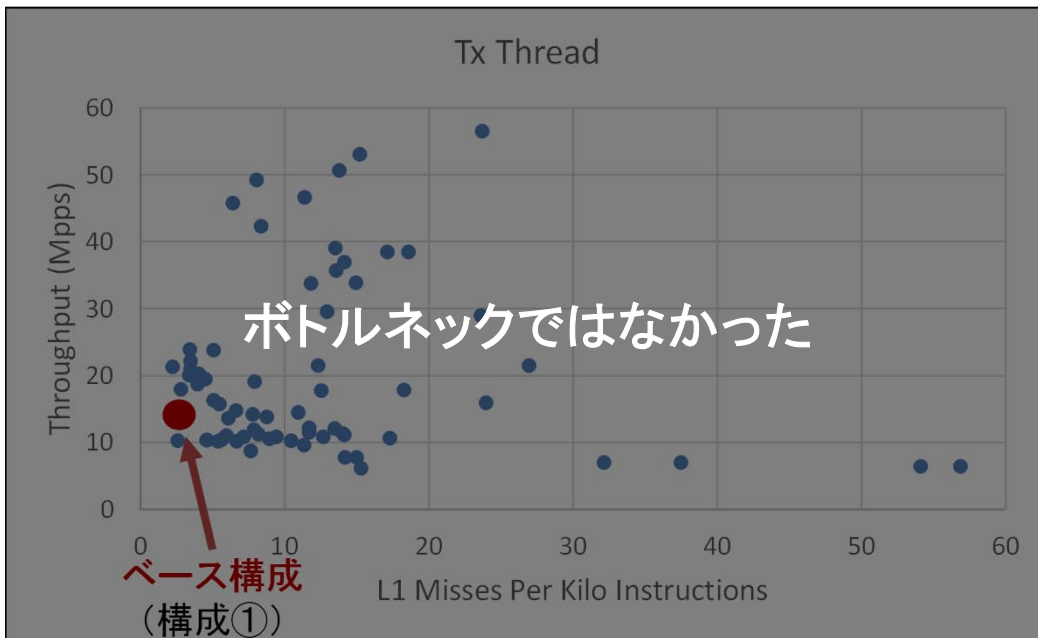
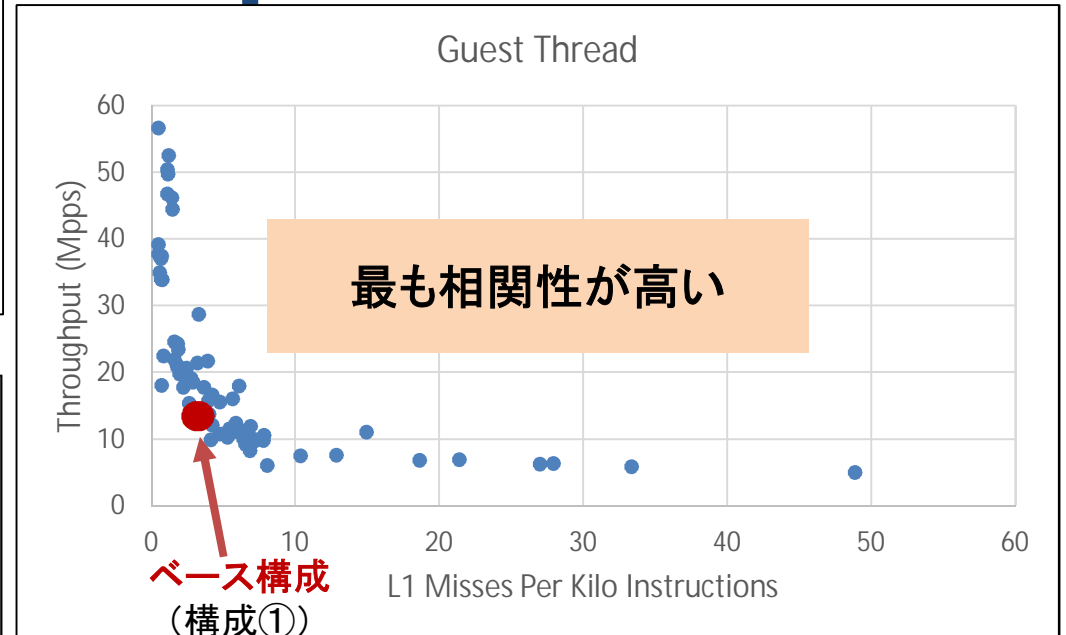
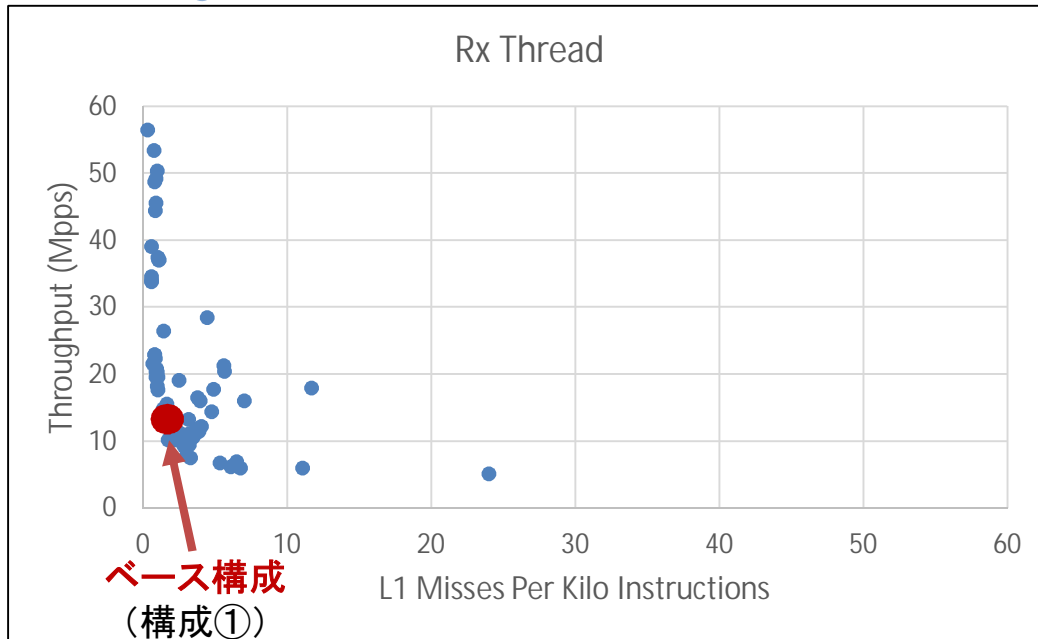


目次

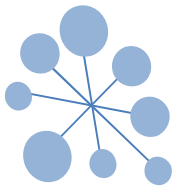
- 研究背景
- 関連研究
- 評価内容
 - 使用するプログラム
 - 評価項目
- **評価結果**
 - キャッシュ利用効率と性能の相関関係
 - 性能向上を実現した要因の特定
 - 仮想I/Oの更なる性能向上に必要な仕様の検討



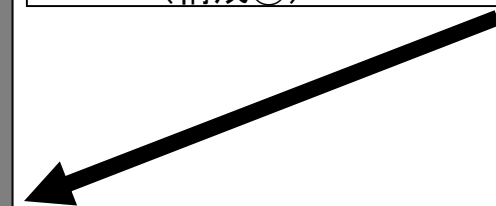
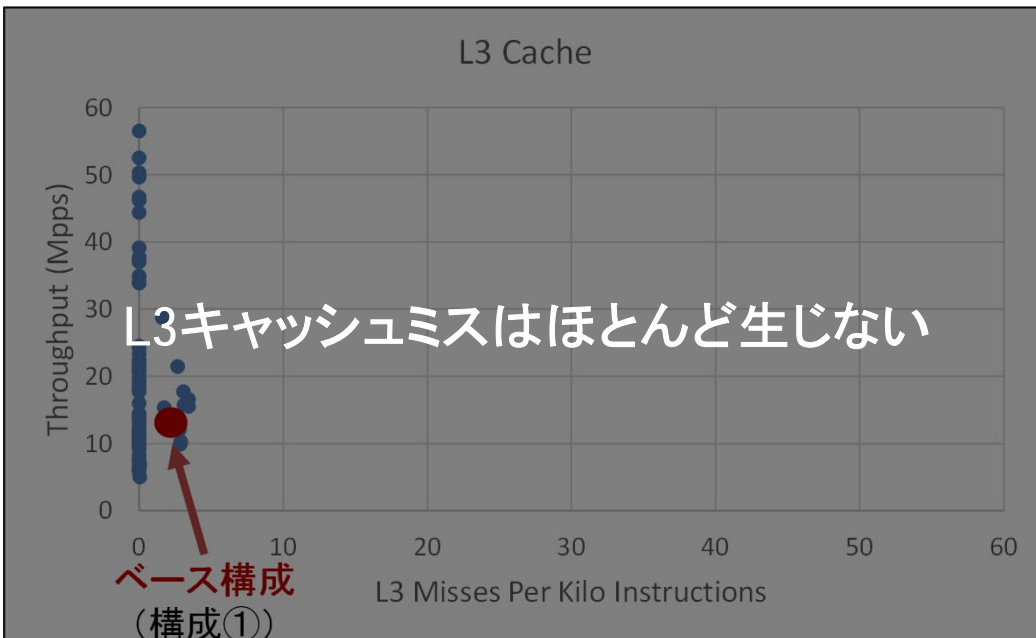
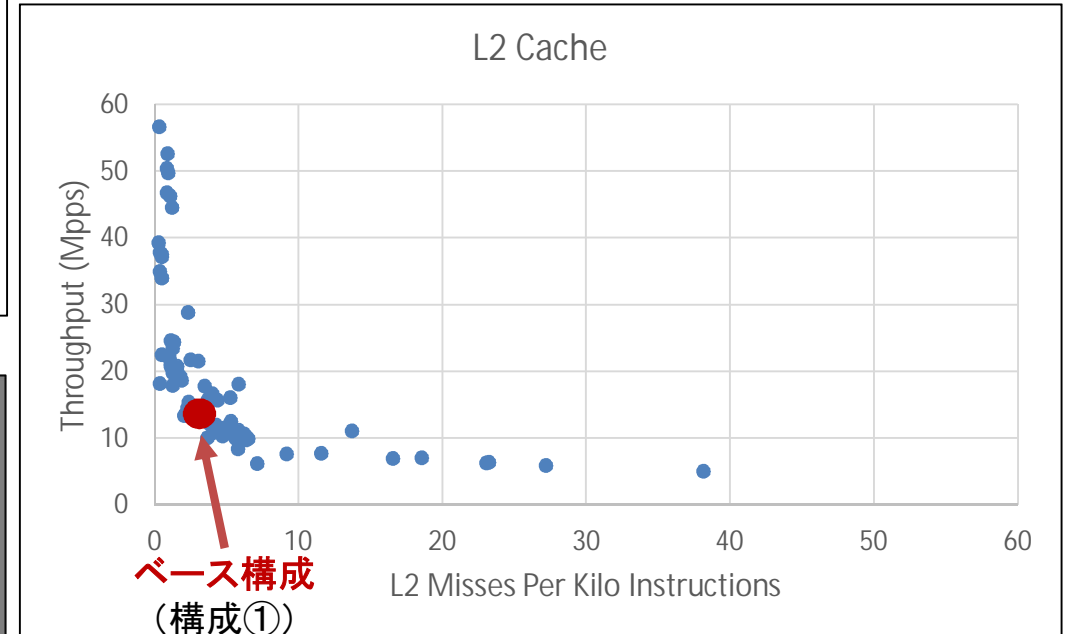
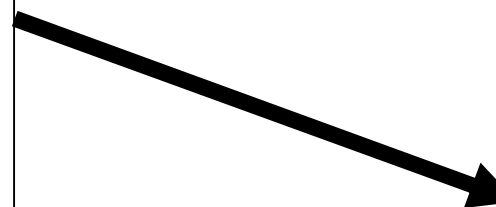
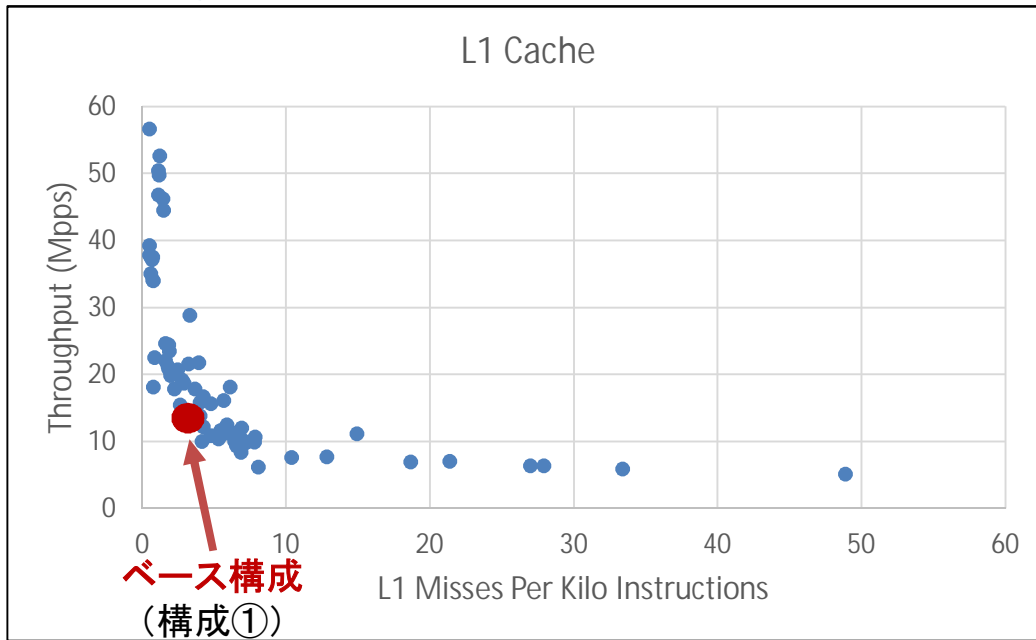
1. 各スレッドにおける相関関係



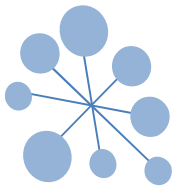
- ・ 4~5倍の性能を実現
- ・ ミス数と性能に指数的な関係



1. L1/L2/L3キャッシュの相関関係

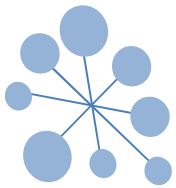


L1, L2キャッシュの利用効率改善が必須



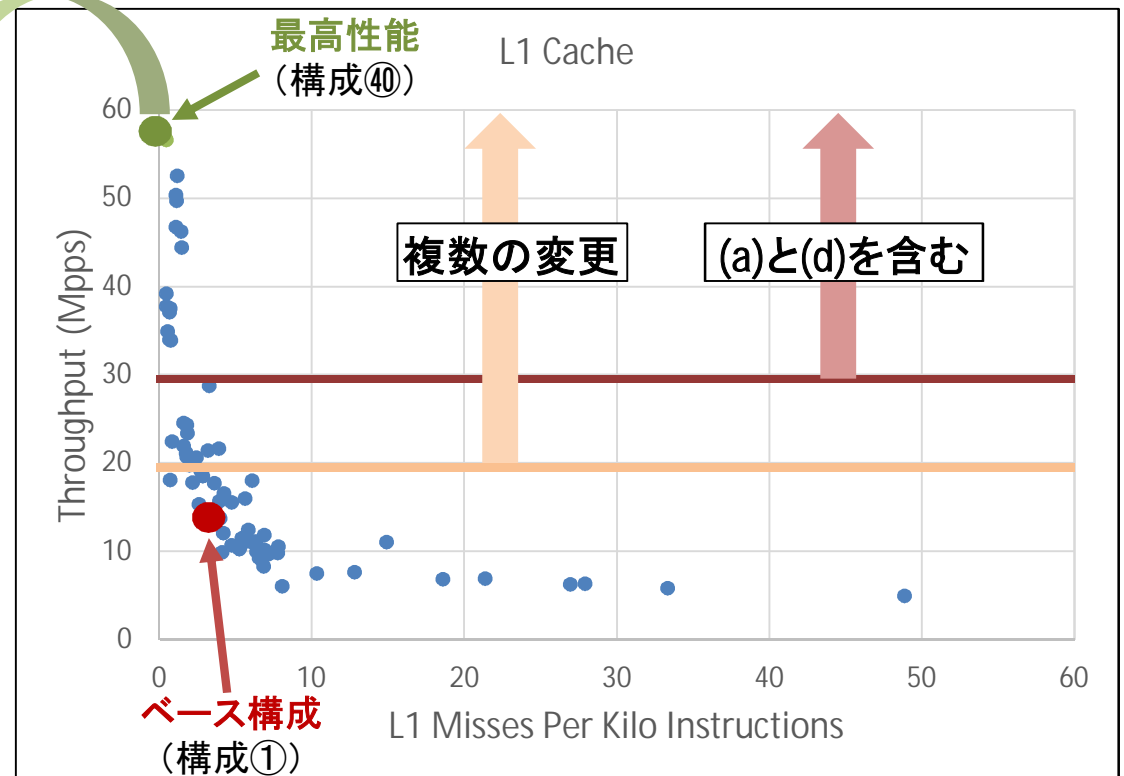
目次

- 研究背景
- 関連研究
- 評価内容
 - 使用するプログラム
 - 評価項目
- **評価結果**
 - キャッシュ利用効率と性能の相関関係
 - 性能向上を実現した要因の特定
 - 仮想I/Oの更なる性能向上に必要な仕様の検討



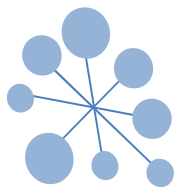
2. 最高性能を実現した因子

因子	概要
(a)	他 CPU コアと共有する変数が全て異なる キャッシュライン上に格納されるように配置
(b)	パケットコピー時に Non-temporal 命令を使用
(c)	Virtqueue における各エントリのサイズを 16B から 8B に縮小
(d)	Mbuf ヘッダのサイズを 128B から 0B に縮小
(e)	パケットデータ格納用領域を 2176B から 64B に縮小
(f)	ハードウェアプリフェッチを無効化



- ・最適化手法の組み合わせが重要
- ・30Mpps以上の実現には(a), (d)が必須

(a), (d)を対象に性能向上の要因を分析



2. (a) による性能向上

同一キャッシュラインに格納
(ベース構成)

```
struct vq {
  int32_t last_avail_idx; // Rxが更新
  int32_t last_used_idx; // Guestが更新
  packed_desc desc[256];
};
```

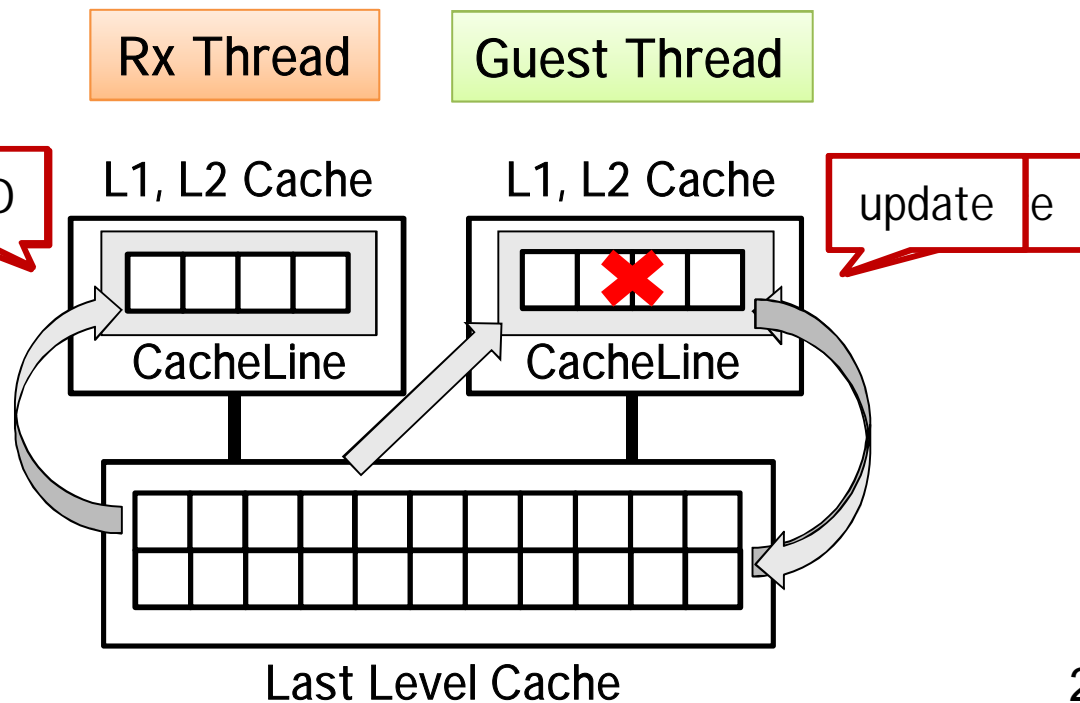
異なるキャッシュラインに格納
(ベース構成 + a)

```
struct vq {
  int32_t last_avail_idx; // Rxが更新
  packed_desc desc[256];
  int32_t last_used_idx; // Guestが更新
};
```

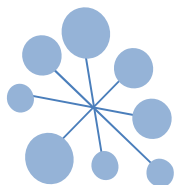
因子	概要
(a)	他 CPU コアと共有する変数が全て異なる キャッシュライン上に格納されるように配置

	base	(a)
L1 Misses / Kilo Instructions	4.1	3.6
L2 Misses / Kilo Instructions	3.6	3.5
Offcore RFO (cycles / packet)	206.4	1.4
Throughput (Mpps)	13.5	17.8

RFO

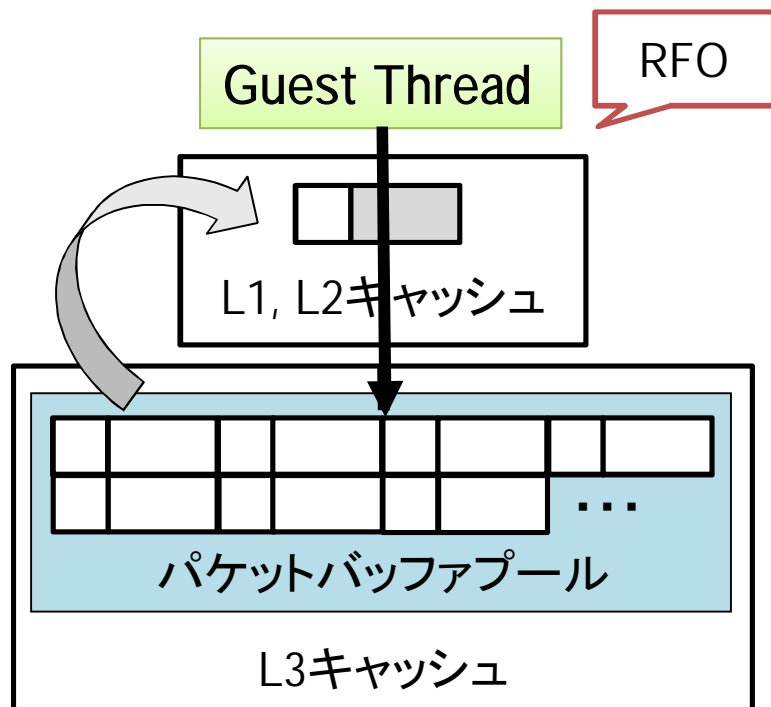


- False Sharingの排除で高速化
- キャッシュ汚染の影響は少ない

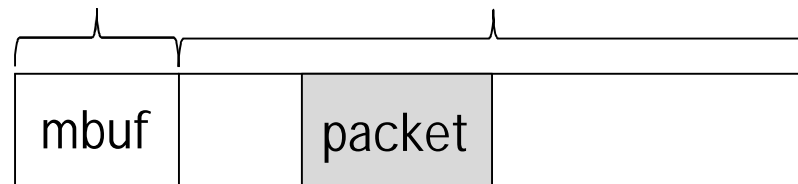


2. (d) による性能向上

因子	概要
(d)	Mbuf ヘッダのサイズを 128B から 0B に縮小



メタデータ領域 パケットデータ領域



- ・ パケットサイズ
- ・ パケットアドレス
- ・ Offload Flag, etc...

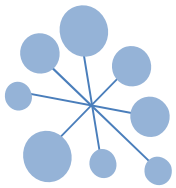
Mbufヘッダ更新の度に

- ・ **RFO**を発行
- ・ **キャッシュミス**

	書き込みのみ	プリフェッチ + 書き込み	書き込まない	(d)
Throughput	13.5 Mpps	17.3 Mpps	20.8 Mpps	20.8Mpps

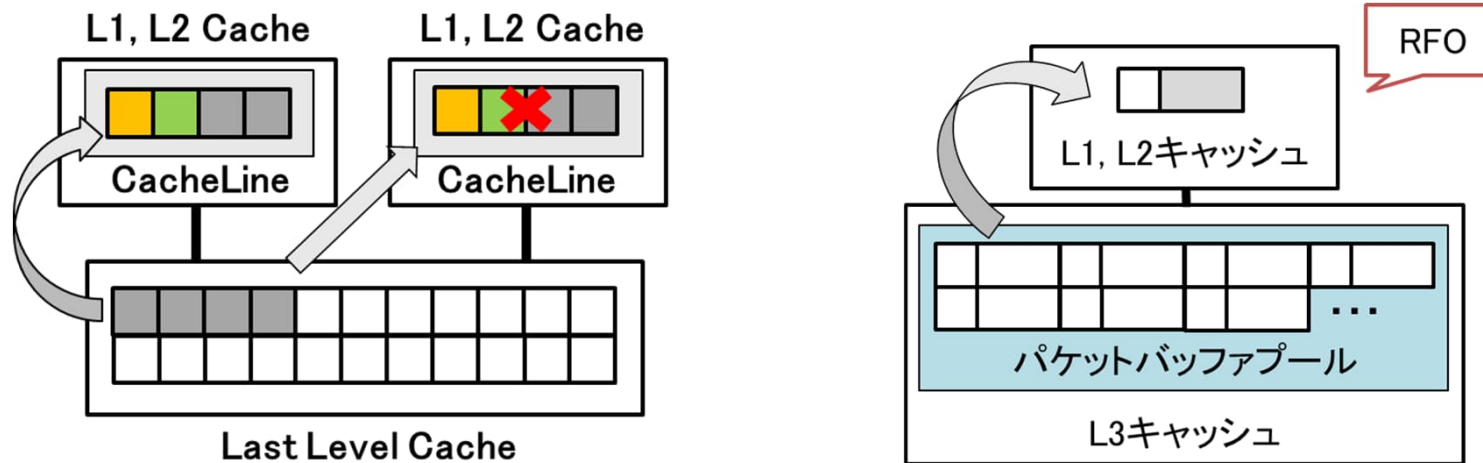
約4Mpps
(キャッシュミス)

約3.5Mpps
(書き込みレイテンシ)



2. (a) と (d) のまとめ

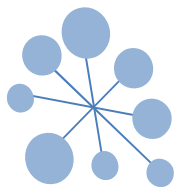
因子	概要
(a)	他 CPU コアと共有する変数が全て異なる キャッシュライン上に格納されるように配置
(d)	Mbuf ヘッダのサイズを 128B から 0B に縮小



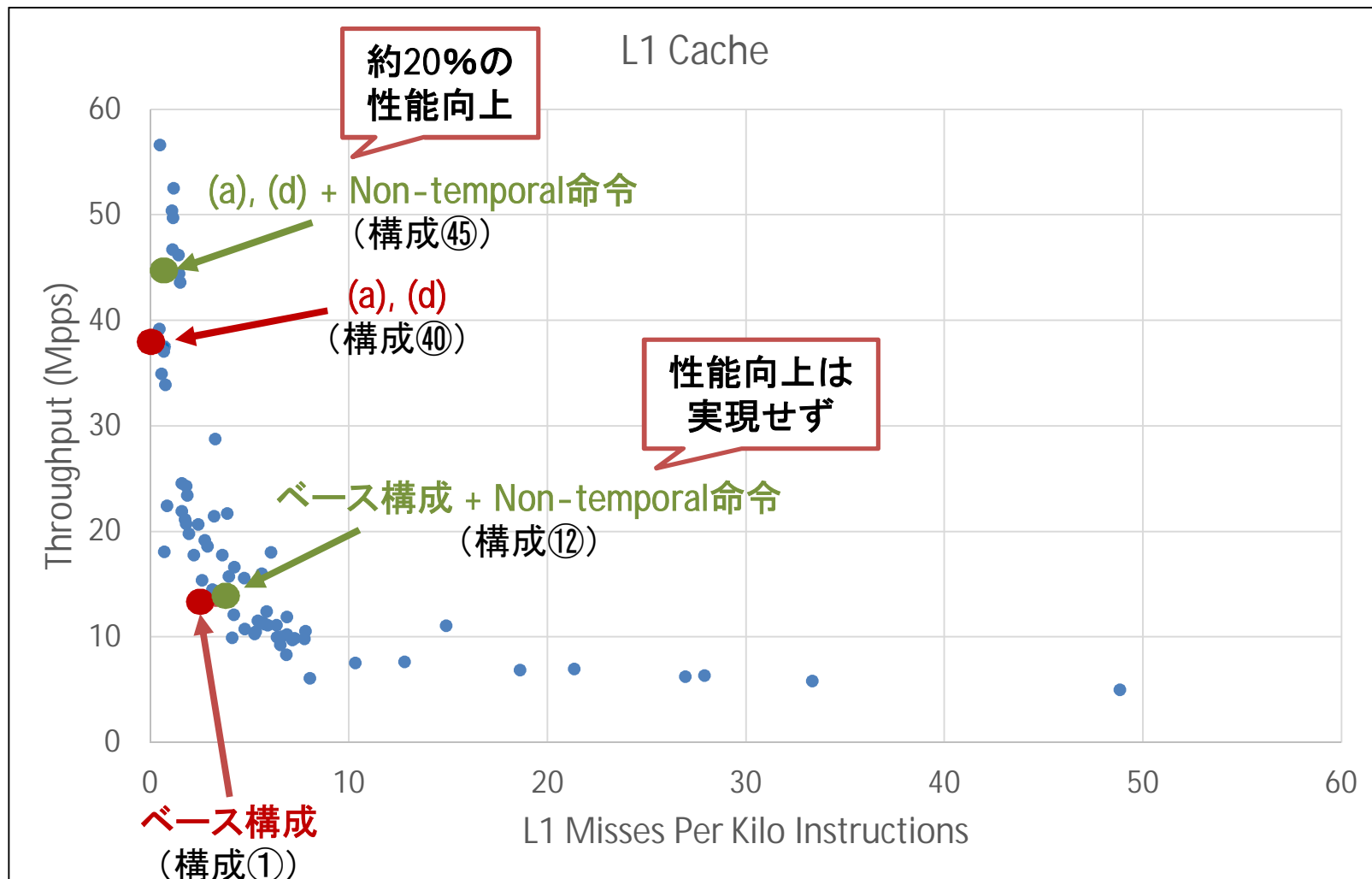
	base	(a)	(d)	(a)&(d)
L1 Misses / Kilo Instructions	4.1	3.6	1.8	0.7
L2 Misses / Kilo Instructions	3.6	3.5	1.1	0.5
Offcore RFO (cycles / packet)	206.4	1.4	2.8	1.5
Throughput (Mpps)	13.5	17.8	20.8	37.5

組み合わせによる
効果が絶大

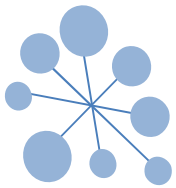
- ・ 他コアとのキャッシュラインの共有を回避
- ・ 書き込みで生じるRFOを抑制



Non-temporal命令による効果の変化

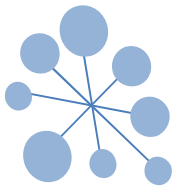


RFOによって性能向上が妨げられていた

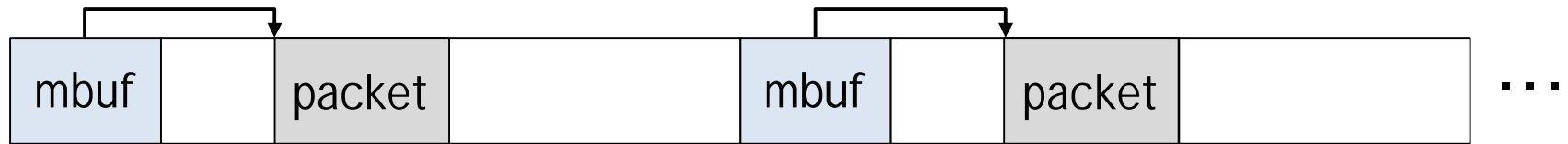


目次

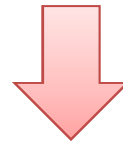
- 研究背景
- 関連研究
- 評価内容
 - 使用するプログラム
 - 評価項目
- **評価結果**
 - キャッシュ利用効率と性能の相関関係
 - 性能向上を実現した要因の特定
 - 仮想I/Oの更なる性能向上に必要な仕様の検討



Mbufヘッダへの書き込みを削減



合計で約10万エントリ

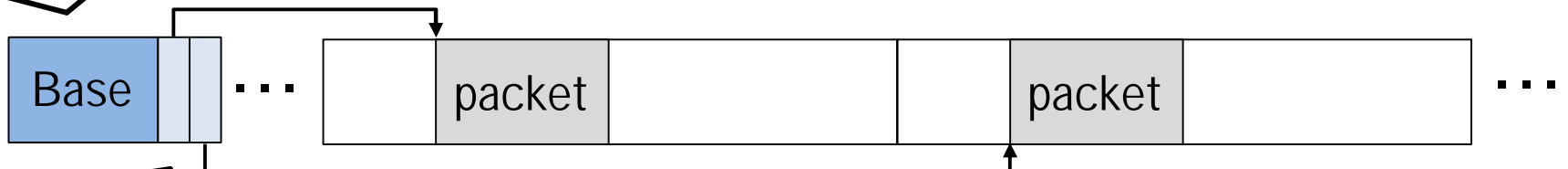


32エントリを再利用

常時キャッシュヒット

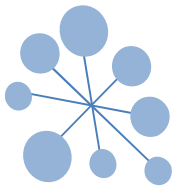


フローで共通の値



バッファ固有の値

RFOの削減



まとめと今後の課題

- vhost-userにおけるキャッシュに着目した調査を実施
 - 性能向上のカギは**キャッシュ**であった
 - Rx, Txスレッドではなく Guestスレッド
 - L1, L2キャッシュミスの削減が重要
 - キャッシュ利用効率の改善により**50Mpps超**の性能を実現
 - アドレス共有による**False Sharing**の排除, **RFO**の排除
 - 複数の組み合わせによる**相乗効果**が重要
 - キャッシュ汚染の解消による性能向上も確認
 - ベース構成では**RFOによる速度低下が大きい**
 - RFOを解消して初めてキャッシュ汚染解消による性能向上がみられる
- **今後の課題**
 - 組み合わせによる相乗効果の考察
 - 性能向上を実現するシステム構成の検討