Online gaming evolution from centralized servers to cloud-based game streaming

- Rise of platforms like Google Stadia, Amazon Luna, and Microsoft Xbox Cloud
- Need for next-gen game engines with modular components (GEMs) for dynamic cloud deployment [1]

Initial co-placement and enhanced player experience (enhance worst case latency slack)

For each placed GEM
- Sort GEMs by latency criticality (maximum tolerable delay)
- Repeat until no gain:
  - For each placed GEM ➔ MOVE: to node that maximizes latency reduction
  - Sort GEMs by experienced latency
  - Repeat until no gain:
    - For each placed GEM ➔ SWAP: with the GEM that maximizes latency reduction

Novel contributions

- Mixed Integer Linear Programming (MILP) formulation for optimal solution (OPT)
- Two-phase heuristic approximations for GEM placement ➔ MAP-MIND and MAP-MIND* (MAnimize Placement and MInimize Delay)
  - MAP (first phase):
    - Sort GEMs by latency critically (maximum tolerable delay)
    - Best-Fit approach for GEM placement
  - MIND (second phase):
    - Sort GEMs by experienced latency
    - Repeat until no gain:
      - For each placed GEM ➔ MOVE: to node that maximizes latency reduction
      - Sort GEMs by experienced latency
      - Repeat until no gain:
        - For each placed GEM ➔ SWAP: with the GEM that maximizes latency reduction

Future work

- Transition from co-located GEMs to a distributed deployment model
- Online placement, enhanced player experience (enhance worst case latency slack)

Published works: 2 journals


Addressed research questions/problems

- Challenges ➔ Shared GEMs, possible complex DAG, maximum tolerable latency, and player distribution
- No past work covered multiple user with shared resources
- No past work considered simultaneously all the above challenges
- How to deploy GEMs in the cloud to maximize session acceptance probability and reduce gaming latency?

GEMs with possible complex DAG:
- Initial co-location ➔ One virtual GEM
- Breaking the problem into two sub-problems:
  - Maximize the acceptance (game provider interest)
  - Minimize the player’s experienced latency (player’s interest)

Adopted methodologies

- An event-driven simulator in Python
- Network scenario ➔ Connected random geometric graph, varying [number of nodes, graph degree, resource utilization factor, heterogeneity of node resources]
- Game scenario ➔ single and multiplayer GEMs, player connection node distribution
- AMPL on CPLEX for the optimal GEM placement.

Comparison:

1. QDH* (extended from Quality Driven Heuristic [2]): Selects GEMs randomly ➔ for each GEM, sorts nodes by delay in an increasing order ➔ applies First-Fit
2. FFD (First-Fit-Decreasing) [3]: Sorts GEMs by resource requirement in decreasing order ➔ finds random eligible node
3. MAP-RNDF (RaNDom First): MAP ➔ for random GEM in placed GEMs ➔ best among first found MOVE, and first found SWAP
4. MAP-RNDG (RaNDom Greedy): MAP ➔ for random GEM in placed GEMs ➔ best among best found MOVE, and best found SWAP
5. MAP-STD (Steepest Descent): MAP ➔ repeat until no gain ➔ Best among best found MOVE, and best found SWAP for all placed GEM
6. RND (RaNDom): Selects GEMs randomly ➔ for each GEM, finds random eligible node

- MAP-MIND: near-optimal latency < 8% blocking in worst-case scenario
- MAP-MIND*: cost-effective and faster, a practical alternative for real-world applications

References

