"A comparative study on Memory Allocators in Multicore and Multithreaded Applications"

Taís B.Ferreira, Rivalino Matias, Autran Macedo, Lucio B. Araujo
Federal University of Uberlândia. Uberlândia – MG, Brazil

Presented by: Luís Miguel Tomé Nóbrega

Introduction

- Running computer programs involves a lot of procedures including many memory allocations and deallocations;
- Memory management is essential for the performance of programs;
- Even more important in Multicore and Multithreads applications;

What is the relevance of this study?

Introduction: Memory allocation

- There are two levels of memory allocation:
  1. **Kernel Level:** memory management of OS sub-systems;
  2. **User Level:** implemented by UMA (user-level memory allocator) that is a library responsible to manage the heap area;

- There is always a default UMA but it can be replaced by another one in order to optimize certain system;

Introduction: choice of the allocator

- Based on experimental tests, often on synthetic benchmarks that perform different and random memory allocation operations;

- Problem: hardly generalized for real applications;

- Solution: this paper presents an alternative way to perform the referred tests, using real applications to perform them.

Allocators: basic operation

Process calls malloc/new

1. Request of a heap area

2. Creation and initialization of the heap header

3. If it is necessary more space, another heap is request (1)

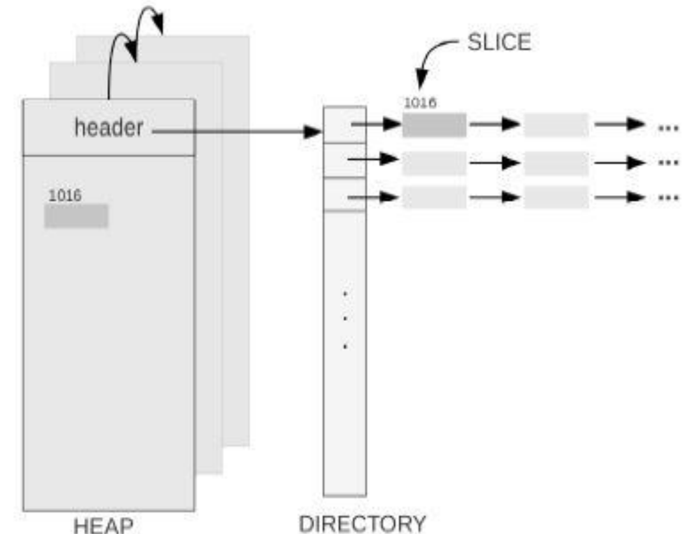

Fig.1:General structures of a UAM

- Data structure (fig.1) is similar to all allocators nevertheless they differ in the way they manage the heap, thus they way they deal with issues like blowup, false sharing and memory contention.

Allocators: Hoard (version 3.8)

- High performance in multithreaded programs running on multicore processors;
- 3 types of heaps:
  1. Thread-cache (<256B)
  2. Local heap;
  3. Global heap;

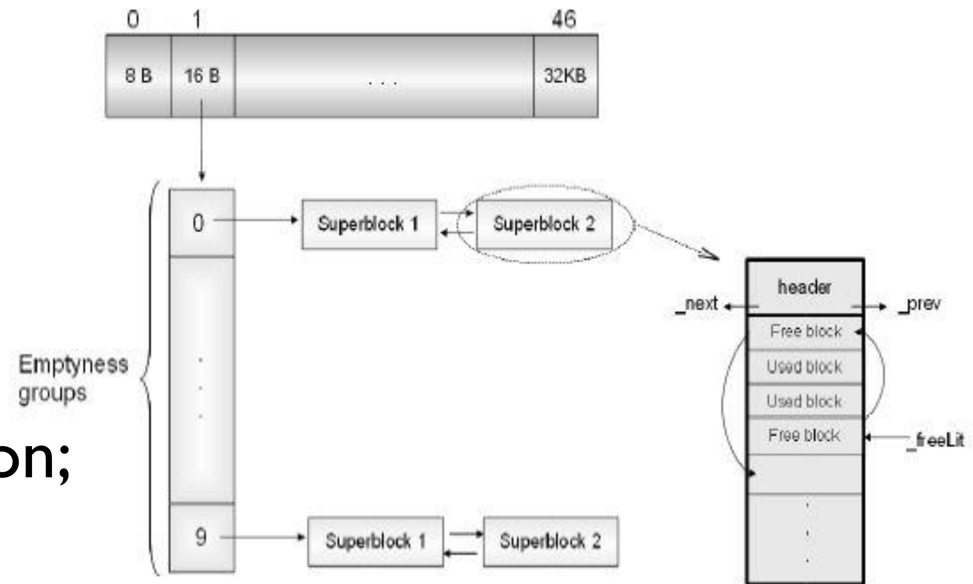

Fig.2: Hoard structures

- Minimize heap contention;
- Minimize blowup;
- Avoids false sharing;

Allocators: Ptmalloc (version 2)

- Also developed for multiprocessors running multithreaded programs;
- Multiple heap areas;
- Does not address false sharing neither blowup;

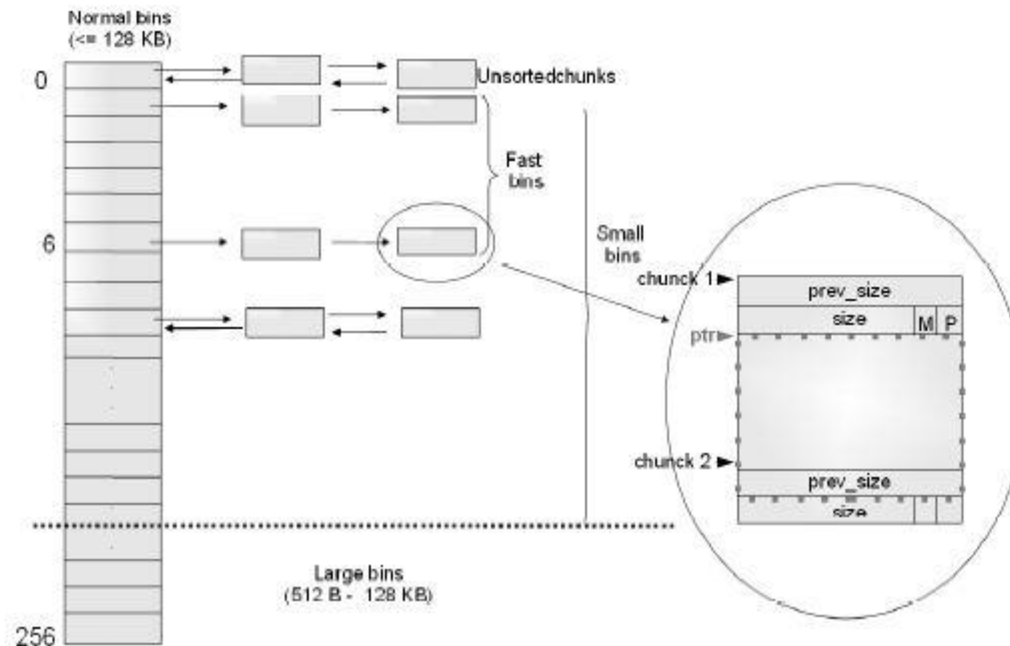

Fig.3: Ptmalloc structures

Allocators: Ptmalloc (version 3)

- Improvement over Ptmallocv2;
- Size of blocks is different;
- Large bins are kept in tree that implements binary search;

Allocators: TCMalloc

- Local heap per thread (from 8B to 32kB);
- Global heap shared by all threads (>32kB);

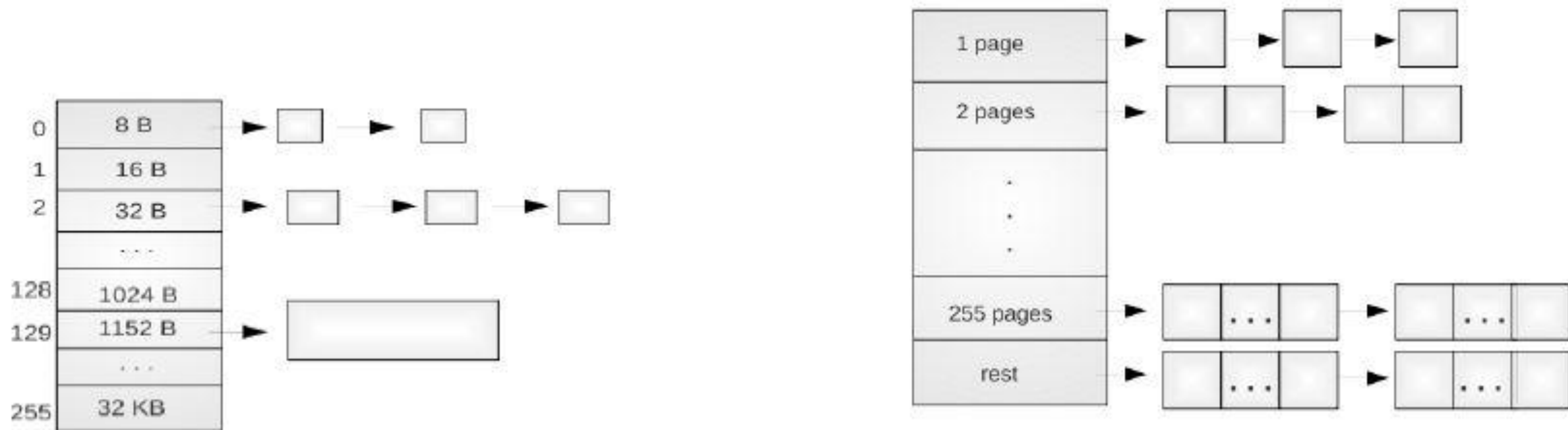

Fig.4: TCMalloc structures

- Minimizes blowup;
- Minimizes contention;
- Does not address false sharing;

Allocators: Jemalloc

- Similar to Hoard, differing in the number of heaps and size of memory objects:
  - Thread cache (<32 kB);
  - Heaps:
    - Small(between 2 and 4kB);
    - large(between 4 and 4MB);
    - huge(>4MB): one shared!!

  Local to threads and 4 per processor! (applies to Small and large)

  - Addresses all the issues.

Allocators: TSLE (version 2.4.6)

- Focused on RT applications;
- Objective is to keep the memory allocation response time constant;
- Unique heap shared by all threads;
- Does not address contention;

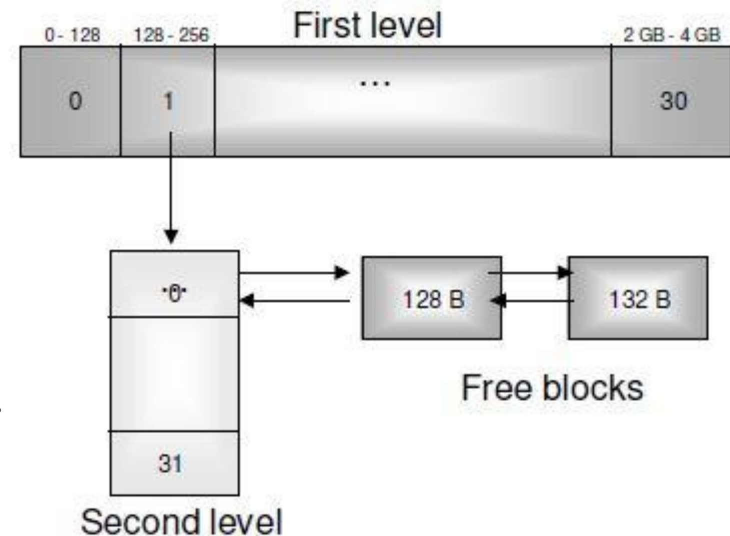

Fig.5 TSLF structures

Allocators: Miser

- Based on Hoard, assumes that the majority of memory requests are up to 256 and try to meet that size fast;
- 1 local heap per thread;
- Global heap;
- Avoid false sharing;
- Avoid blowup;
- Does not solve requests for memory blocks larger then 256B;

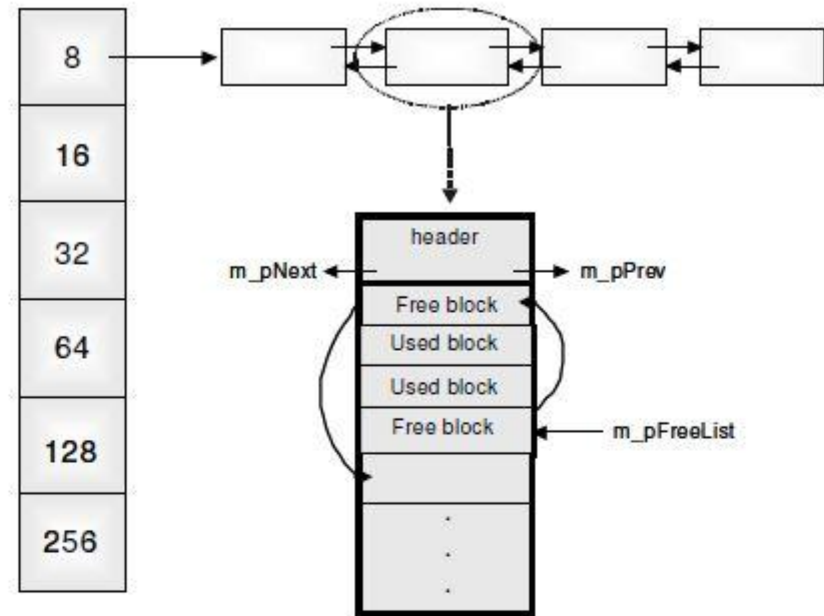

Fig.6 Miser structures

Experimental study

- Methodology:
  1. Characterization of the memory usage in each application;
  2. Linking applications to each allocator and analyze performance in terms of response time, memory consumption and memory fragmentation;

- Instrumentation
  - 3 Core Duo 2.4 GHz, 2 – gigabyte Ram
    1. Running the middleware applications (3 applications);
    2. Database;
    3. Workload generator tools;

Result Analysis

- Request size distribution

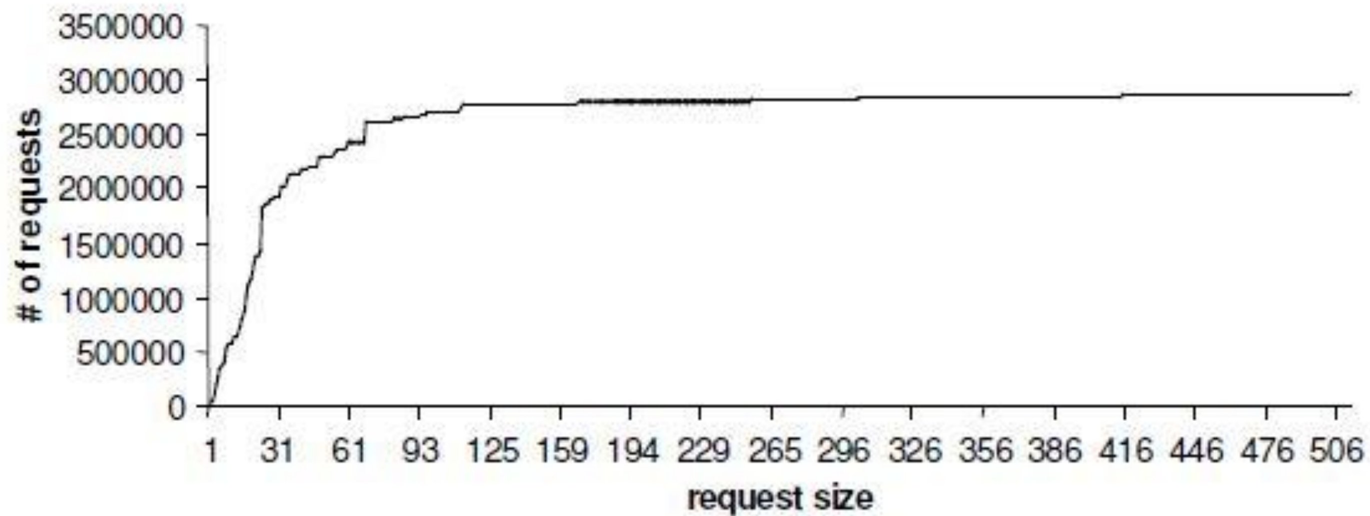

Fig.7 Request size distribution

Result Analysis

- Middleware performance per allocator

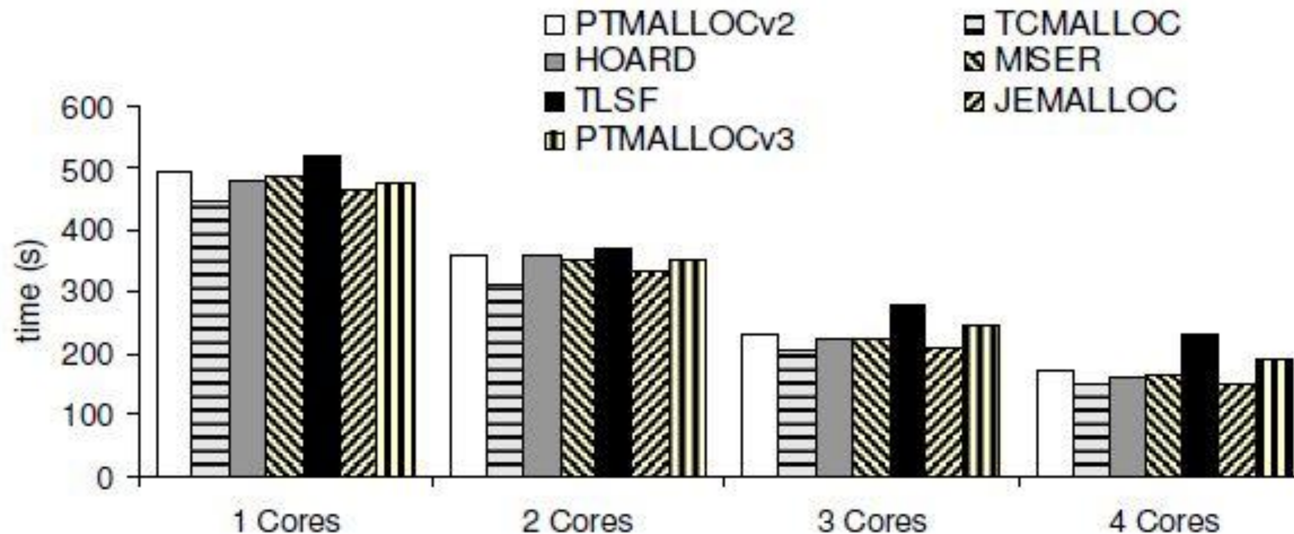

Fig.8 Middleware performance per allocator

Result Analysis

- Memory consumption per allocator

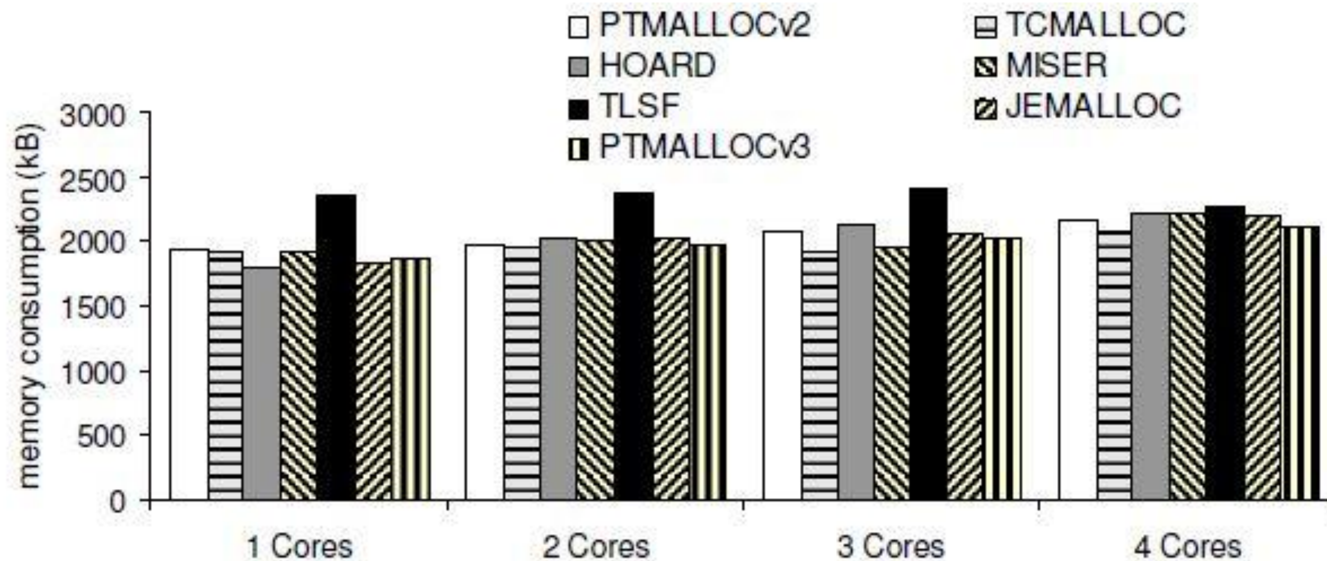

Fig.9 Memory consumption per allocator

Result Analysis

- Fragmentation level per allocator

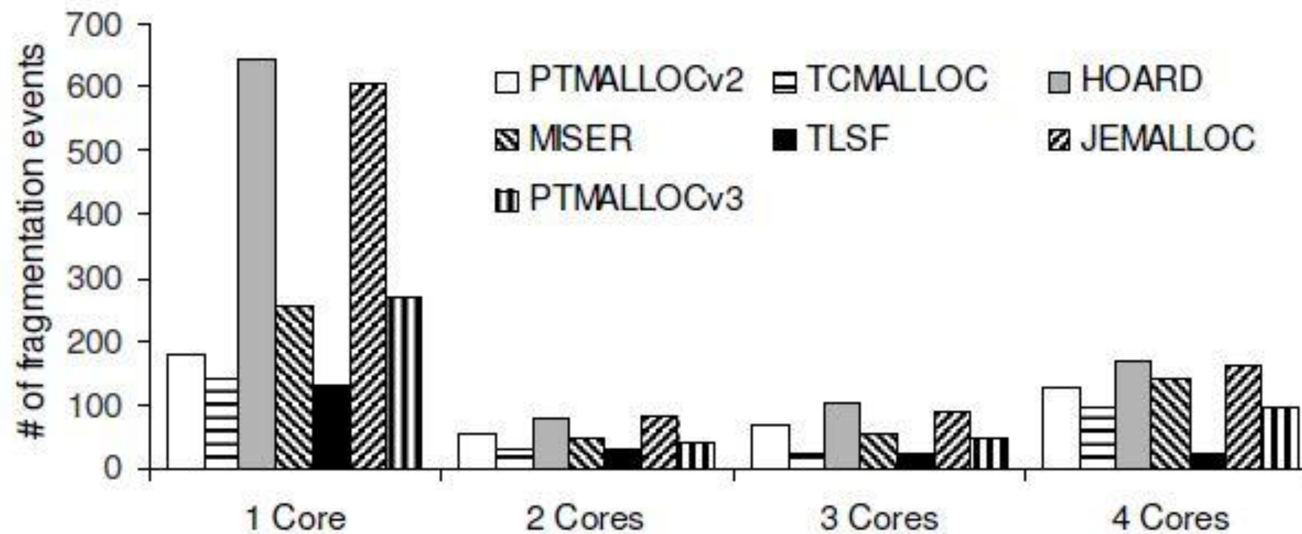

Fig.9 Fragmentation level per allocator

Conclusions

- TCMalloc presents the best results followed by Ptmallocv3;
- Jemalloc and Hoard show very good performance in terms of response time but high memory consumption and fragmentation;