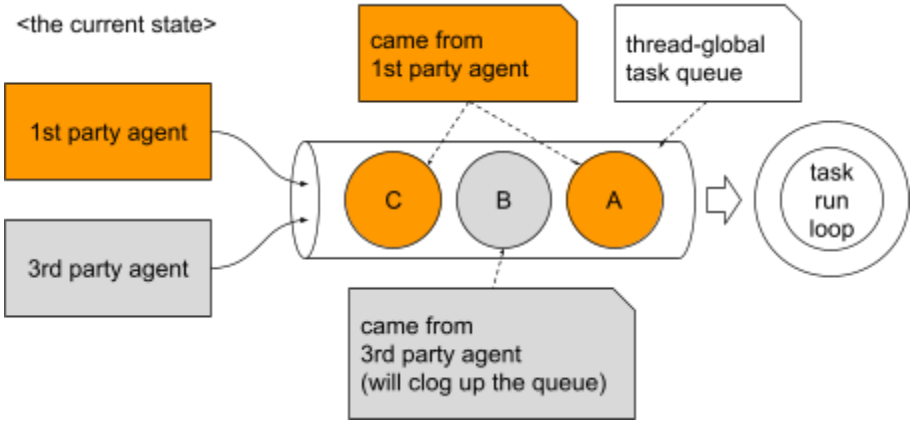
PUBLIC

[Design Doc] MBI:AgentGroupScheduler

chikamune@
Status: draft
Tracking bug: https://crbug.com/1105403
Last Update: 2020-07-30

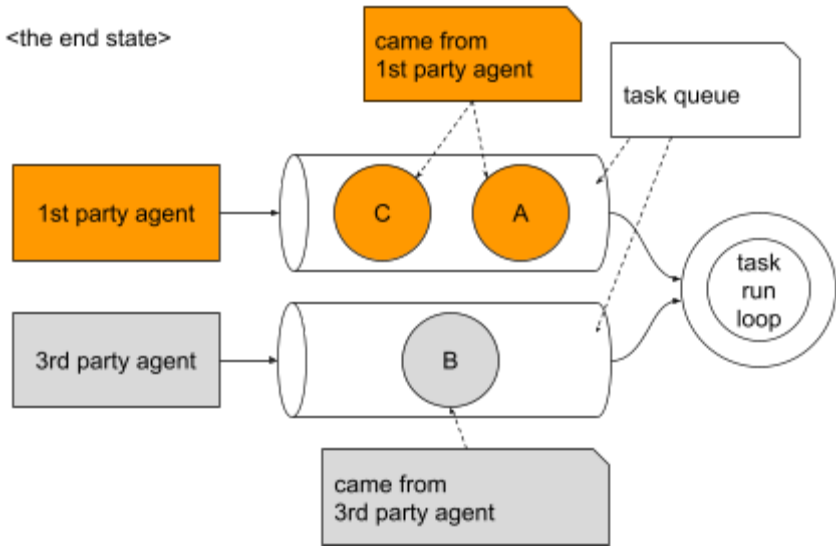
Motivation

The “MBI: Per-isolate Scheduling” project wants to prioritize 1st party agents and deprioritize 3rd party agents. Currently, agents share the same thread-global task queues. This means that a thread-global task queue contains tasks from both 1st and 3rd party agents. On the premise that a task can’t overtake the previous tasks inside the same task queue, we can’t prioritize only 1st party agents. Because of this a task from the 3rd party agent will clog up the queue. We call this problem the “task ordering problem”. To solve this problem, we are going to have separate task queues per Isolate.



Goals

- Split the task queues for the 1st party agents from the task queues for the 3rd party agents. This design doc is intended to achieve this MVP.



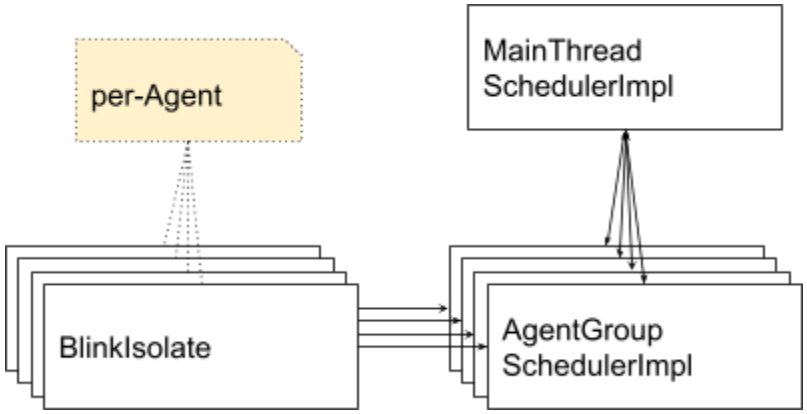
Non-goals

These items are out of scope.

- Cleaning up the scheduler code
- Worker support
- Per-isolate heap
- Multi-Main Threads (MMT)

How AgentGroupScheduler solves task ordering problem

An agent has a group of Frames which can be synchronously scriptable to each other. AgentSchedulingGroup will be created per agent if the device has enough memory. An AgentGroupScheduler will be created for each AgentSchedulingGroup. If we can split the current thread-global task queues into per-AgentSchedulingGroup task queues, the 1st party agents and 3rd party agents will use different task queues. Then the task ordering problem will be solved.



The thread-global task queues

There are 10 thread-global task queues. I'm going to migrate some of these task queues to AgentSchedulingGroup. It is clear that we need to keep control_task_queue on MainThreadSchedulerImpl because control_task_queue is used by the scheduler itself. Probably the other queues can be moved to AgentGroupScheduler.

task queue	migrate to AgentGroupScheduler?	reason / comment
default_task_queue	yes	The default task queue will be used for general purpose.
control_task_queue	no	The scheduler hierarchy uses this queue. This queue will be accessed from the compositor thread.
ipc_task_queue	merge to per-isolate default_task_queue	We are going to get rid of this queue, and merge this queue to default_task_queue.
idle_queue	yes	IdleHelper uses this queue to run low priority idle tasks. And this queue is used from many places. But it is only accessed from the main thread.
cleanup_task_queue	merge to per-isolate default_task_queue	This is used from RenderThreadImpl, RenderViewImpl and WidgetBase to shut down things. We are going to merge this queue to per-isolate default_task_queue.

non_waking_task_queue	no	This is used by MemoryUsageMonitor which periodically checks the memory usage and notifies its observers.
compositor_task_queue	yes	This queue is used for compositing task. So this queue has to be per-isolate.
v8_task_queue	no	v8_task_queue is used for GC. When we move v8::Isolate per-AgentSchedulingGroup during per-isolate heap project, we want to move v8_task_queue to per-AgentSchedulingGroup.
memory_purge_task_queue	no	This is used by MemoryPurgeManager. MemoryPurgeManager manages process-wide proactive memory purging. We shouldn't stop this queue.
virtual_time_control_task_queue	not mandatory	This is used by Web Rendering Service (headless browser). So this will not cause the task ordering problem.

Deprecation of ipc_task_queue

The scheduling policy of ipc_task_queue is the same as the default task queue. Nowadays, we are using per-frame task queues as much as possible. And if there are no relevant per-frame task queues, then we can use the (per-isolate) default task queues. [discussion thread]

Deprecation of cleanup_task_queue

The scheduling policy of cleanup_task_queue is the same as the default task queue. We can simply use the (per-isolate) default task queue instead of cleanup_task_queue. [discussion thread]

The task queue containers

Currently, MainThreadSchedulerImpl holds thread-global task queues, and FrameSchedulerImpl holds per-frame task queues.

MainThread SchedulerImpl	Page SchedulerImpl	Frame SchedulerImpl
default_task_queue control_task_queue ipc_task_queue idle_queue cleanup_task_queue non_waking_task_queue compositor_task_queue v8_task_queue memory_purge_task_queue virtual_time_control_task_queue	(doesn't have any task queues)	contains a lot of per-frame task queues on HashMap

We are going to introduce AgentGroupScheduler to hold per-isolate task queues. Normally, AgentGroupScheduler will be created per-agent. By switching AgentGroupScheduler, we can switch per-isolate task queues.

MainThread SchedulerImpl	MainThreadIsolate SchedulerImpl	Page SchedulerImpl	Frame SchedulerImpl
control_task_queue	default_task_queue ipc_task_queue idle_queue cleanup_task_queue non_waking_task_queue compositor_task_queue v8_task_queue memory_purge_task_queue virtual_time_control_task_queue	(doesn't have any task queues)	contains a lot of per-frame task queues on HashMap

Threading

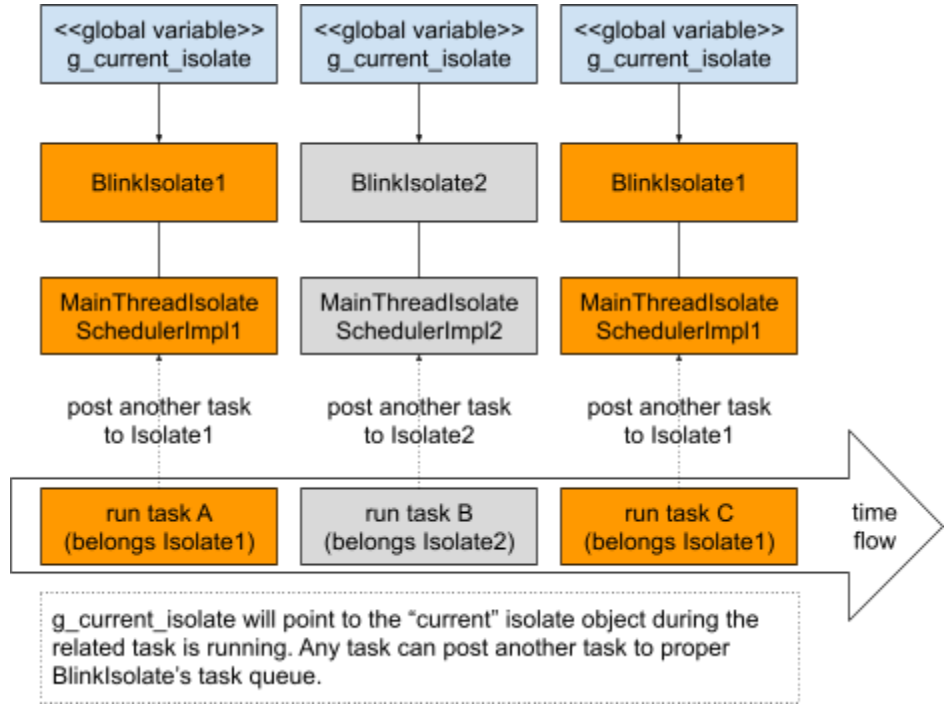
The control_task_queue is accessed from many threads. But we are not going to migrate control_task_queue to Isolate level (at least from the beginning). And all the task queues except the control_task_queue is accessed from the main thread. When we migrate the thread-global task queues (and the task runners) to AgentGroupScheduler, AgentGroupScheduler will be accessed only from the main thread.

Task queue / Task runner caching

Many code caches task queues and task runners. Most of the cache doesn't have any problem because most of the caches are bound to proper single Isolate. But if there is a cross-isolate caching, we have to modify such cache so that the cache returns a proper object.

How to post new tasks to proper per-Isolate task queues

The code which needs to know the current AgentSchedulingGroup will run in the main thread. Other threads don't have any idea of AgentSchedulingGroup. So we can store the current AgentSchedulingGroup object to g_current_isolate global variable. In this way, we can know which AgentSchedulingGroup and AgentGroupScheduler should be used at this time via this g_current_isolate global variable. AgentGroupScheduler will have its own task queues and task runners for the Isolate. So we can post a new task to proper Isolate's task queues.



ExecutionContext::GetIsolate

For example, when the main thread is running task A which belongs to Isolate1, the current g_current_isolate will point to AgentSchedulingGroup1. Then task A can know the proper AgentGroupScheduler via g_current_isolate, and task A can post a new task to the proper task queue on AgentGroupScheduler1. When the SequenceManager chose task B as a next task which belongs to Isolate2, then g_current_isolate will automatically point to AgentSchedulingGroup2, and task B can post a new task to the proper task queue on AgentGroupScheduler2. In this way, any task can post new tasks into proper task queues. Specifically, we can get the default_task_queue with this code below.

```
AgentSchedulingGroup::GetCurrent()->Scheduler()->DefaultTaskQueue()
```

So we need to update all the callsites which use the current thread-global task queues (and the task runners) with this kind of code.

How to update the g_current_isolate

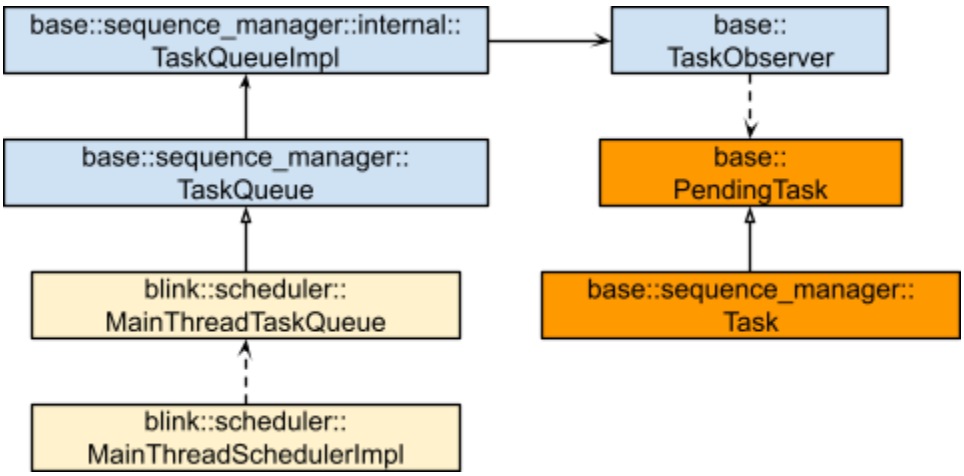
The g_current_isolate global variable should point to the current AgentSchedulingGroup. SequenceManagerImpl::SelectNextTask() function automatically decides the next WorkQueue, and takes a Task from the WorkQueue. So we need to update g_current_isolate based on the automatically scheduled next Task.

How to know the execution timings

TaskObserver is a good place to update the g_current_isolate global variable. TaskObserver has two hook functions to know the task execution timings.

```
class BASE_EXPORT TaskObserver {
 public:
  virtual void WillProcessTask(const PendingTask& pending_task,
                               bool was_blocked_or_low_priority) = 0;
  virtual void DidProcessTask(const PendingTask& pending_task) = 0;
};
```

This TaskObserver can be used in several ways. In our case, we can attach TaskObserver to TaskQueue so that it can be noticed when the task (which is taken from that TaskQueue) is going to be processed.



How to know/update the next AgentSchedulingGroup

We can simply create a TaskObserver implementation below. And we can attach AgentSchedulingGroupTaskObserver to per-isolate task queues and per-frame task queues. But we shouldn't attach AgentSchedulingGroupTaskObserver to thread global task queues because there is no proper isolate in the thread global task queues.

```
class AgentSchedulingGroupTaskObserver : public TaskObserver {
 public:
  AgentSchedulingGroupTaskObserver(AgentSchedulingGroup* blink_isolate);
  void WillProcessTask(const PendingTask& pending_task,
                       bool was_blocked_or_low_priority) override {
    AgentSchedulingGroup::GetCurrent() = blink_isolate_;
  }
  void DidProcessTask(const PendingTask& pending_task) override {
    AgentSchedulingGroup::GetCurrent() = nullptr;
  }

 private:
  AgentSchedulingGroup* blink_isolate_;
};
```

Another idea to update the g_current_isolate

There is another good place to update the g_current_isolate global variable. MainThreadSchedulerImpl has OnTaskStarted() and OnTaskCompleted() hook points.

```
void OnTaskStarted(
    MainThreadTaskQueue* queue,
    const base::sequence_manager::Task& task,
    const base::sequence_manager::TaskQueue::TaskTiming& task_timing);

void OnTaskCompleted(
    base::WeakPtr<MainThreadTaskQueue> queue,
    const base::sequence_manager::Task& task,
    base::sequence_manager::TaskQueue::TaskTiming* task_timing,
    base::sequence_manager::LazyNow* lazy_now);
```

From these API, we can get MainThreadTaskQueue objects. MainThreadTaskQueue has GetFrameScheduler() function, and FrameSchedulerImpl will have a pointer to AgentGroupScheduler. So we can obtain the next AgentSchedulingGroup. This way is easier than the idea of TaskObserver. Attaching TaskObserver to per-frame task queues was difficult.

How to confirm if the destination is correct

There is a concern about the task posting destination. When we are processing a task in a specific isolate, we should post a new task to the same isolate. To confirm this, we might be able to add some checks on the PostTask function. [TBD]

Thread global default task runners

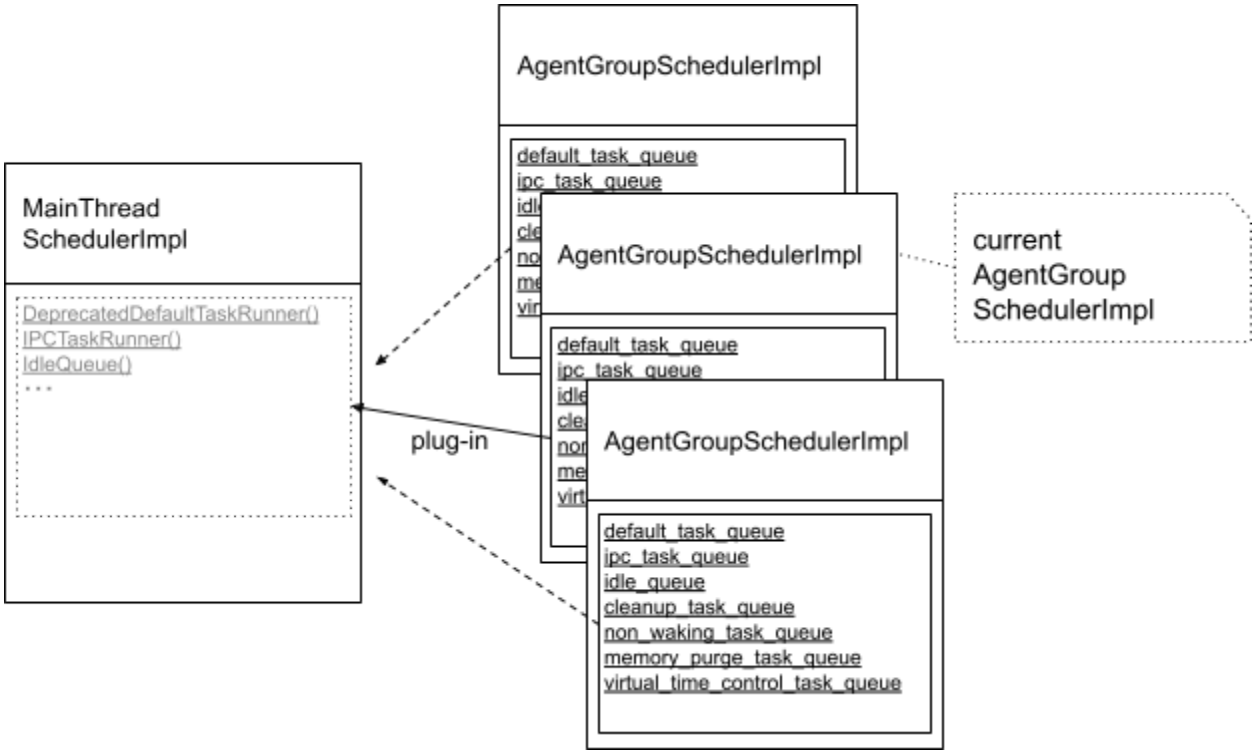
There are several ways to get the thread global default task runner.

- base::ThreadTaskRunnerHandle::Get()
- base::SequencedTaskRunnerHandle::Get()

These functions are called from not only the main thread but also other threads. We need to modify these functions so that it returns the correct isolate's task runner when it's called from the main thread. If these functions are called from other than the main thread, it must behave as it is before. When these functions are called from the main thread, we need to get the current AgentSchedulingGroup via AgentSchedulingGroup::GetCurrent(). If AgentSchedulingGroup::GetCurrent() returns non null AgentSchedulingGroup, these methods will return per-isolate default task runners. If AgentSchedulingGroup::GetCurrent() returns nullptr, these methods will return null.

[optional] Intermediate state during the migration

In the end, all the call sites which use the current thread-global task queues have to be migrated. But in the beginning, we want to keep the current logic in MainThreadSchedulerImpl as it is because we want to keep the CLs small. To do this, we will modify task queue getter functions and task runner getter functions in MainThreadSchedulerImpl. These getter functions will return the "current" Isolate task queues or task runners implicitly. This implicit migration makes it easy to know which task_queue has problems or not. When we confirm that everything works fine, then we want to remove these implicit getter functions from MainThreadSchedulerImpl.



Code cleaning up

When we migrate the task queues and task runners from MainThreadSchedulerImpl to AgentGroupScheduler, it might be a good idea to move some logics from MainThreadSchedulerImpl to AgentGroupScheduler if the logic is closely coupled with the task queues or task runners.

Action Items

Landing schedule of the CLs is listed below.

#	Status	Action Item
1	Done	Introduce an empty AgentGroupScheduler.
2	Done	Merge ipc_task_queue to default_task_queue.
3	Done	Merge cleanup_task_queue to default_task_queue.
4	WIP	Introduce switching mechanism of AgentGroupScheduler. [ref]

5	WIP	Migrate default_task_queue to AgentGroupScheduler's AgentGroupScheduler.
6	Blocked	Integrate AgentGroupScheduler with AgentSchedulingGroup.