

10 - SequentialCoordinator

Iagro CLP Local — SequentialCoordinator

Runtime **padrão** do CLP Local: desacopla leitura OPC (alta frequência) de tarefas serializadas (DB, fila, captura receita) para reduzir corrida em `write_commands` e carga no PostgreSQL.

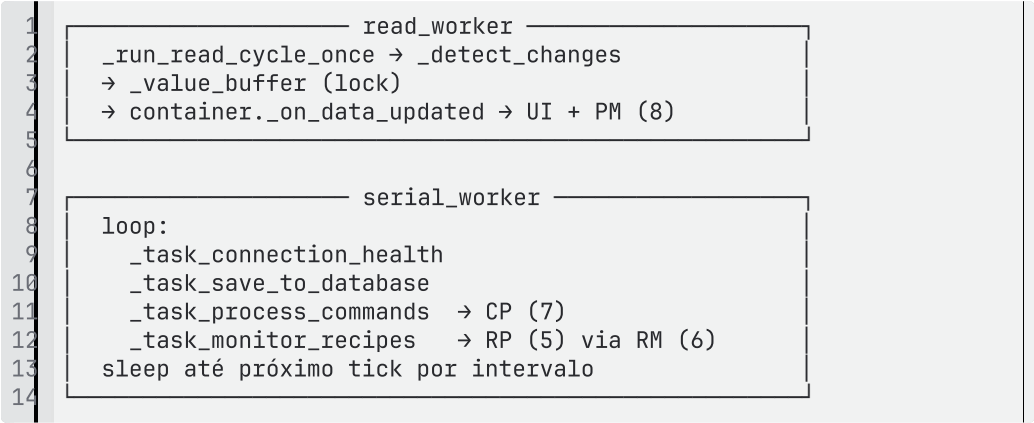
Código: `src/core/sequential_coordinator.py`

Resumo

Thread	Função
<code>read_worker_thread</code>	Ciclo OPC contínuo → buffer → <code>on_data_updated</code>
<code>serial_worker_thread</code>	Health, persistência, comandos, receitas

`CLPContainer.monitor_thread` aponta para `serial_worker_thread`.

Arquitetura dupla



Loop serial (`_run_loop`) — tarefas

Ordem	Tarefa	Intervalo típico	Implementação
0	Health OPC	~2000 ms	<code>_task_connection_health</code> — detecta queda,

			mark_connect ion_lost
1	Flush buffer leituras	DB_WRITE_INT ERVAL_MS (~200)	_task_save_t o_database
2	Fila comandos	COMMAND_POLL _INTERVAL_MS	_task_proces s_commands
3	Captura receita	2000 / 10000 adaptativo	_task_monito r_recipes

Métricas: `container.metrics.record_serial_loop(duration)` .

`apply_performance_intervals(container.config)` mapeia env → atributos do coordenador na subida do monitor.

`_task_process_commands` (detalhe)

```

1 1. check_executing_recipe_parents(container, command_processor)
2   └─ sanitize_orphan_create_recipes (fim da função)
3 2. SE NOT container.is_connected(): RETURN
4 3. pending ← get_pending_write_commands(limit=10, container_id)
5 4. PARA cmd IN pending:
6   command_processor.execute_command(cmd)
```

Diferença vs CP thread: batch **10** (coordenador) vs **100**

(`CommandProcessor.process_commands` em modo thread).

Pré-requisito: `command_processor` existe e **não** está em `run()` paralelo — `_sync_command_processor_thread` garante.

Pós-reconexão

Health worker pode chamar `check_executing_recipe_parents` sem processor após reconnect — reconcilia pais `executing` antes do próximo poll de comandos.

`_task_monitor_recipes`

Replica gates do **6 - RecipeMonitorService**:

Gate	Fonte
CLP + DB	callbacks / <code>database_manager</code>

<code>has_pending_recipe_commands</code>	DB
<code>get_active_recipe</code>	ApiService
<code>has_pending_reset</code>	ProductionMonitor

API: `get_work_order_recipe_sync(status=NA_FILA, limit=1)` — uma candidata por tick.

```
recipe_processor.process_recipe(recipe, is_connected).
```

🔄 Leitura OPC (`_read_worker_loop`)

- Prioridades: env `OPCUA_PRIORITY_*` + ids de `get_active_view_variables` (supervisório).
- Mudanças enfileiradas no buffer; serial worker persiste em lote.
- `on_data_updated` dispara na thread de leitura — UI usa scheduler thread-safe.

⚙️ Variáveis de ambiente (via `container.config`)

Chave	Efeito
<code>DB_WRITE_INTERVAL_MS</code>	Frequência flush histórico
<code>COMMAND_POLL_INTERVAL_MS</code>	Poll fila
<code>RECIPE_POLLING_FAST_MS</code> / <code>SLOW</code> / <code>ADAPTIVE_TIMEOUT</code>	Captura NA_FILA

⚠️ Modo paralelo (contraste)

Aspecto	Com coordenador	Paralelo
<code>CommandProcessor.run</code>	Off	Thread ativa
<code>RecipeMonitorService.run</code>	Off	Thread ativa
Leitura	<code>read_worker</code>	<code>VariableMonitorThread</code> + AutoTuner

Corrida fila	Mitigada (serial + SQL filtro)	Depende de intervalos
--------------	--------------------------------	-----------------------

Referências

- **9 - CLPContainer** (`_sync_command_processor_thread`)
- **7 - CommandProcessor** · **6 - RecipeMonitorService**
- **4 - ApiService** (`get_active_view_variables`)