Plan de correction — réduire le CPU MySQL maintenant

Application **150M-Client** (Laravel 10.50.2) · Base **reach** · Objectif : baisser rapidement l'usage CPU mysqld (~80 % sous charge) avec des correctifs ordonnés par ROI.

ORDRE D'EXÉCUTION RECOMMANDÉ — (1) Aujourd'hui : cache Schema::hasTable/hasColumn + pause des polls onglet caché + ralentir nearby/notifications/activités. (2) Cette semaine : FAB count cache, historique paginé SQL, batch GPS planning cart. (3) Ensuite : stats SQL GROUP BY, carte unités snapshot. Ne commencez PAS par ajouter des index seuls : le PROCESSLIST montre surtout information_schema (tempête Schema::hasTable).

1. Diagnostic confirmé (production)

Preuve	Observation	Cause code	Impact CPU
SHOW FULL PROCESSLIST	Plusieurs SELECT information_schema.tables (276 tables, ~1106 rows examined) en parallèle	Schema::hasTable() appelle getTables() = scan COMPLET du schéma à chaque appel (~1621 call sites)	Très élevé sous concurrence
PROCESSLIST	information_schema.columns sur zone_scopes	Schema::hasColumn('zone_scopes', …)	Élevé (répété)
PROCESSLIST	organisation_node_unit_territories unit 628	Planning / carte / contraintes territoire	Faible isolé
Audit layout	Polls 8s / 12s / 15s / 5s GPS sur chaque onglet	app.blade.php + nearby + live-position + activités 5s	Élevé (volume)
Audit pages	Historique load-all, stats ->get(), N+1 carte	VisitHistoryAggregator, PerformanceAnalytics, planningCart	Très élevé à l'ouverture

2. Roadmap en 3 vagues

Vague	Délai	Actions	Gain CPU estimé	Risque
VAGUE A — MAINTENANT	Aujourd'hui / 1–2 jours	Cache schéma + throttle polls + pause document.hidden + nearby adaptatif	Fort et immédiat (metadata + volume HTTP)	Faible
VAGUE B — COURT TERME	Cette semaine	FAB count cache, historique SQL paginé, batch GPS cart, activités 1 WHERE IN	Fort sur pages terrain	Moyen (tests)
VAGUE C — MOYEN TERME	1–2 semaines	Stats GROUP BY + cache, snapshot carte unités, fusion notifications, Redis présence	Stabilise sous charge haute	Moyen

3. VAGUE A — Correctifs immédiats (faire maintenant)

A1. Cache Schema::hasTable / hasColumn (priorité absolue PROCESSLIST)

Laravel 10 n'a pas de commande schema:cache fiable pour hasTable. Chaque Schema::hasTable() relit information_schema.tables (toutes les tables). Correctif : memoization par requête HTTP + cache applicatif (fichier/Redis) des flags connus.

```
// AppServiceProvider::boot() — idée de correctif
use Illuminate\Support\Facades\Schema;
use Illuminate\Support\Facades\Cache;

// 1) Memoization request-level (évite N scans dans le même request)
$tablesMemo = [];
$columnsMemo = [];

// 2) Wrapper helper recommandé (à utiliser dans les hot paths) :
// schema_has_table('cible_positions')
// qui lit Cache::remember('schema.tables', 3600, fn() => Schema::getTableListing())
// puis in_array, au lieu de Schema::hasTable en boucle.

// Alternative rapide : patcher via Macro / service SchemaCache
// et remplacer progressivement les hot paths :
// NearbyVisitCibleDetectorService, FieldUnitActivityService,
// PendingVisitsForUnitService, VisitRuntimeService, PlanningJournalierController.
```

Étape A1	Action concrète	Fichiers / zone
1	Créer App\Support\SchemaCache (hasTable/hasColumn memo + Cache 1h)	app/Support/SchemaCache.php
2	Remplacer Schema::hasTable dans hot paths (nearby, FAB, activités, planning)	Services Visits/Dashboard + PlanningJournalierController
3	Invalider le cache après migrations (hook migrate ou artisan schema-cache:clear)	AppServiceProvider / commande
4	Vérifier PROCESSLIST : disparition des SELECT information_schema.tables en rafale	Production

A2. Freiner les polls du layout (réduction volume HTTP → moins de SQL)

Poll	Actuel	Nouveau (immédiat)	Où changer	Gain
Nearby visit	15 s	30–45 s + pause si document.hidden + skip si GPS peu bougé (<20 m)	nearby-visit-prompt.blade.php / .js	÷2 à ÷3 appels terrain
Notifications feed	8 s	20–30 s + pause hidden	layouts/app.blade.php pollMs	÷2.5 à ÷4
Impact modal	12 s	Fusionner avec le feed OU 30 s	script inline app.blade.php	Supprime doublon
Activités du jour	5 s	20–30 s + pause hidden	dashboard/activites-du-jour.blade.php POLL_MS	÷4 à ÷6 sur cette page
Live GPS push	~5 s	10–15 s et seuil 15–20 m	live-position-tracker.js	Moins d'écritures
Stats / commandes	30 s	60–120 s	engine.blade.php / statistiques.blade.php	Moins d'agrégations

```
// Pattern JS à ajouter sur TOUS les setInterval de polling
document.addEventListener('visibilitychange', function () {
  if (document.visibilityState === 'hidden') { clearInterval(timer); timer = null; }
  else { pollOnce(); timer = setInterval(pollOnce, POLL_MS); }
});
```

A3. Nearby — ne plus scanner à vide

Même avec un intervalle plus long, chaque detect() reste lourd. Correctifs immédiats côté JS + PHP :

#	Correctif	Détail
1	Pause onglet caché	Aucun POST /portal/nearby-visit-check si hidden
2	Intervalle adaptatif	45 s si immobile ; 15 s seulement après déplacement > 20–30 m
3	Cache serveur 20–30 s	Cache::remember(userId+celluleGPS, 25s) du résultat detect()
4	Enrichir 1 seul candidat	Favoris / planning / panels uniquement pour nearest, pas pour 25
5	SchemaCache	Zéro information_schema dans la boucle nearby

A4. Checklist déploiement Vague A (aujourd'hui)

■	Tâche	Validation
■	Déployer SchemaCache + remplacer hot paths critiques	PROCESSLIST : plus de rafales information_schema.tables
■	pollMs notifications 8000 → 25000	Network tab : ~2.4 req/min/user au lieu de ~7.5
■	Impact poll 12000 → 30000 ou fusionné	1 seul endpoint notifications
■	Nearby 15000 → 30000 + hidden + move threshold	Moins de POST nearby
■	Activités POLL_MS 5000 → 25000 + hidden	Page ouverte : ÷5 charge
■	Mesurer CPU mysqld 15 min avant/après	Objectif : baisse nette du %CPU au même nombre d'onglets

Gain attendu Vague A : souvent 30–60 % de baisse du trafic SQL répétitif + disparition du bruit information_schema. Suffisant pour soulager un CPU à 80 % si la charge vient surtout des onglets ouverts.

4. VAGUE B — Correctifs court terme (cette semaine)

B1. FAB pending — ne plus scanner ~20 tables pour un badge

Aujourd'hui listForUser() tourne au rendu Blade (chaque navigation) + poll 180 s. Remplacer par un compteur cache.

```
// fab-pending-unit-visits.blade.php — NE PLUS FAIRE :
$count = count(app(PendingVisitsForUnitService::class)->listForUser($u, 50));

// FAIRE :
$count = Cache::remember("fab.pending.{$clientId}.{$unitId}", 45, function () use ($u) {
    return app(PendingVisitsForUnitService::class)->countForUser($u); // COUNT SQL léger
});
// Endpoint poll : renvoyer seulement { count: N }
// Liste détaillée : uniquement à l'ouverture de l'offcanvas.
```

B2. Historique visites — pagination SQL (stop load-all)

VisitHistoryAggregator fait ->get() sur toutes les tables puis paginate en PHP. Corriger avec fenêtre de dates par défaut (30–90 j) + LIMIT/OFFSET (ou keyset) par table, ou une table de projection visit_history_index alimentée à l'écriture.

B3. Planning cart — batch GPS / détails

Remplacer loadEntityGps() et cartEntityDetailRows() dans la boucle foreach stops par : grouper les IDs par entity_type → 1 SELECT WHERE IN par type. Différer les détails au clic pin.

B4. Activités du jour — 1 passe multi-unités

Remplacer foreach ($childUnitIds as $cid) { collectTodayItemsForUnitIds([$cid]); } par un seul collectTodayItemsForUnitIds($allUnitIds) puis groupBy unit_id en PHP. Batch OrganisationNode names (plus de N+1 coaching).

Correctif B	Effort	Gain	Tests minimaux
FAB count cache	1–2 j	Élimine ~20 SELECT / navigation terrain	Badge correct ; offcanvas charge la liste
Historique SQL paginé + dates défaut	2–4 j	Page stable même historique long	Filtres, export chunked
Planning cart batch	1–3 j	SELECT ≈ types au lieu de ≈ arrêts	Carte pins OK, détail au clic
Activités 1 WHERE IN	1–2 j	÷N enfants sur le poll	Manager multi-unités ; poll 25 s

5. VAGUE C — Moyen terme (stabilisation)

Correctif	Quoi faire	Pourquoi
Stats / performances	Remplacer ->get() + agrégation PHP par COUNT/SUM/GROUP BY SQL. Table aggregates_daily. Cache Redis par onglet+période. Refresh 60–120 s.	Périodes 30–90 j explosent le CPU
Carte unités/réseau	Snapshot JSON cache 30–60 s par manager (unités + assignments). Sortir Schema::* des boucles.	SSR unit×cible trop cher
Notifications unifiées	1 endpoint {unread, max_id, new_items, modal?} toutes les 20–30 s. ETag / 304. Plus de COUNT à chaque poll.	3 consommateurs de la même table
Live positions	UPSERT natif ; optionnel Redis pour positions éphémères ; persist toutes les 30–60 s.	Réduit binlog / writes
Index (après EXPLAIN)	Valider puis ajouter seulement les index manquants (voir section 7). Jamais en premier.	Index sans réduire le volume = gain limité

6. Actions ops immédiates (sans code)

Action	Commande / réglage	Effet
Mesurer avant	SHOW GLOBAL STATUS LIKE 'Questions'; SHOW GLOBAL STATUS LIKE 'Com_select';	Baseline
Top digests	performance_schema.events_statements_summary_by_digest ORDER BY SUM_TIMER_WAIT DESC LIMIT 20	Confirmer information_schema vs métier
Slow log temporaire	long_query_time=0.5 ou 1 ; digérer 10–15 min de pic	Capturer patterns
Limiter connexions idle trop longues	wait_timeout raisonnable ; pool PHP-FPM/Laravel	Moins de Sleep inutiles
Ne pas redémarrer MySQL comme « fix »	Traite le symptôme ; les polls reviennent en 1 minute	Inutile seul

7. Index candidats (après Vague A, avec EXPLAIN ANALYZE)

N'ajoutez ces index qu'après confirmation EXPLAIN. Un index redondant augmente le coût d'écriture (live GPS, nearby, présence).

```
-- Notifications
(target_client_user_id, client_id, id)
(target_client_user_id, client_id, read_at, id)

-- PF / nearby
client_pf_memberships (client_id, entity_type, is_active, entity_id)
cible_positions (org_key, cible_id)
cible_working_day_positions (org_key, cible_id, day)

-- Territoires (vu dans PROCESSLIST)
organisation_node_unit_territories (organisation_node_unit_id, cible, commune_id)

-- Activités / historique
visites_* (client_id, organisation_node_unit_id, started_at)
visites_* (client_id, organisation_id, started_at)  -- selon filtres stats

-- Live
client_user_live_positions UNIQUE(client_user_id)
(organisation_id, organisation_node_unit_id, updated_at)
```

8. Plan de mesure (prouver la baisse CPU)

Moment	Mesure	Succès
T0 avant	%CPU mysqld, Questions/s, Com_select/s, PROCESSLIST 3×	Baseline écrite
Après A1 SchemaCache	PROCESSLIST sous charge	0 ou très rares information_schema.tables
Après A2–A3 polls	Access log : req/min sur nearby, feed, activités	÷2 à ÷5 selon poll
Après Vague B	Temps /visites/historique, /planning/cart, activités	p95 en baisse nette
Pic réel 50+ onglets	%CPU mysqld	Objectif < 40–50 % au même trafic utilisateur

9. Ce qu'il NE faut PAS faire en premier

Anti-pattern	Pourquoi
Ajouter 20 index « au feeling »	N'arrête pas information_schema ni les polls 5–15 s ; peut freiner les writes GPS
Scaler uniquement le VPS	Le trafic SQL croît linéairement avec les onglets ; le coût revient
Couper MySQL / redémarrer	Soulage 2 minutes ; les setInterval repartent
Optimiser une page rare avant SchemaCache + polls	ROI faible vs correctifs globaux
Désactiver nearby complètement sans alternative	OK en urgence 24 h, pas une solution produit

10. Fichiers à toucher (carte d’implémentation)

Priorité	Fichier	Changement
A1	app/Support/SchemaCache.php (nouveau) + AppServiceProvider	Cache hasTable/hasColumn
A1	NearbyVisitCibleDetectorService, FieldUnitActivityService, PendingVisits…, VisitRuntimeService, PlanningJournalierController	Utiliser SchemaCache
A2	resources/views/layouts/app.blade.php	pollMs notif 25s ; impact 30s ou fusion
A2	resources/views/layouts/partials/nearby-visit-prompt.blade.php + public/js/nearby-visit-prompt.js	30–45s + hidden + move
A2	resources/views/dashboard/activites-du-jour.blade.php	POLL_MS 25000 + hidden
A2	public/js/live-position-tracker.js	10–15s / 15–20 m
A2	public/js/portal-notifications-feed.js	Respecter visibilityState
B1	fab-pending-unit-visits.blade.php + PendingVisitsForUnitService	count cache ; liste à l’ouverture
B2	VisitHistoryAggregator + VisitesController::historique	Pagination SQL + dates défaut
B3	PlanningJournalierController::planningCart	Batch GPS/détails
B4	FieldUnitActivityService::buildPollPayload	1 collect multi-unités
C	PerformanceAnalyticsService, DashboardUnitesReseauCarteController, notifications	Agrégats / snapshot / feed unique

11. Synthèse « que faire maintenant » (ordre strict)

```
1) Implémenter SchemaCache et brancher les hot paths → vérifier PROCESSLIST.
2) Monter les intervalles de poll + pause document.hidden (layout + activités + nearby).
3) Cache résultat nearby 20–30 s + enrichissement du seul nearest.
4) Déployer, mesurer CPU 15 min.
5) Enchaîner FAB count + historique paginé + batch planning cart.
6) Ensuite seulement stats SQL et index confirmés par EXPLAIN.
```

Focus unique pour aujourd’hui : SchemaCache + frein des polls. C’est le couple d’actions le plus aligné avec votre PROCESSLIST réel et le plus rapide à livrer pour faire redescendre le CPU mysqld.

Documents liés : docs/rapport-requetes-layout-polling-mysql.pdf · docs/rapport-pages-requetes-lourdes-mysql.pdf